NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

UNSTEADY CASEWALL PRESSURE MEASURMENTS IN A TRANSONIC COMPRESSOR

by

William R Levis

June 2006

Thesis Advisor:	Garth Hobson
Second Reader:	Anthony Gannon

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Unsteady Casewall Pressure Measurements in a Transonic Compressor during Steam Induced Stall			**5. FUNDING NUMBERS**
6. AUTHOR(S) William R Levis			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

During launch of aircraft off of a carrier deck, steam leakage is sometimes ingested into the aircraft's engine and may cause a compressor stall or "pop-stall". As the US Navy prepares to field the single engine F-35C Joint Strike Fighter, it becomes necessary to investigate the phenomenon known as "pop-stall". In the present study, steady-state as well as transient measurements prior to and during a steam induced rotating stall were taken. Changes to the honeycomb altered the performance characteristics of the Transonic Compressor Rig and needed to be remapped in order to determine a new stall line as well as a peak performance criterion. Data was taken at 90 percent design speed as well as during a 70 percent steam induced stall with the aide of 9 Kulites at varying positions along the case wall. Data was reduced and analyzed through the use of a data acquisition and data reduction system.

14. SUBJECT TERMS Compressor, Transonic, Steam Ingestion, Turbulence, Stall			**15. NUMBER OF PAGES** 111
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

UNSTEADY CASEWALL PRESSURE MEASURMENTS IN A TRANSONIC COMPRESSOR

William R. Levis
Ensign, United States Navy
B.S., United States Naval Academy, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2006

Author: William R. Levis

Approved by: Prof. Garth Hobson
Thesis Advisor

Dr. Anthony Gannon
Second Reader

Dr. Anthony Healey
Chairman
Department of Mechanical and Astronautical Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

During launch of aircraft off of a carrier deck, steam leakage is sometimes ingested into the aircraft's engine and may cause a compressor stall or "pop-stall". As the US Navy prepares to field the single engine F-35C Joint Strike Fighter, it becomes necessary to investigate the phenomenon known as "pop-stall". In the present study, steady-state as well as transient measurements prior to and during a steam induced rotating stall were taken. Changes to the honeycomb altered the performance characteristics of the Transonic Compressor Rig and needed to be remapped in order to determine a new stall line as well as a peak performance criterion. Data was taken at 90 percent design speed as well as during a 70 percent steam induced stall with the aide of 9 Kulites at varying positions along the case wall. Data was reduced and analyzed through the use of a data acquisition and data reduction system.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Professor Hobson for the amount of effort and time that he put in to help me complete my thesis. I would also like to thank John Gibson and Rick Still for their hard work and quick wit making the Turbopropulsion Laboratory a more enjoyable place to work. I have appreciated my time here and couldn’t have asked for better people or a better place to work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

As the United States Navy begins to transition to the F-35C Joint Strike Fighter (JSF) it becomes necessary that the phenomenon known as a "pop-stall" be resolved. A "pop-stall" occurs when the catapult-launch system on an aircraft carrier releases steam during a launch cycle. The steam is than ingested into the intakes of the aircraft, causing a fan or compressor stall and a possible total engine stall or "pop-stall". Experiments conducted at Naval Air Engineering Station Lakehurst with an F-18 demonstrated the relative susceptibility of the aircraft to "pop-stall" events. This susceptibility of the dual engine F-18 is of significant interest because as the Navy begins to transition to the single engine F-35C, the probability of a "pop-stall" occurring and causing a catastrophic loss of an aircraft increases.

The work done at the Turbopropulsion Laboratory (TPL) at the Naval Postgraduate School (NPS) focuses on the "pop-stall" problem with the use of the Transonic Compressor Rig (TCR). The transonic compressor fan stage was specifically designed for the Naval Postgraduate School to be used in the TCR by Sanger (1996) at the NASA Glenn Research Center (Ref. 1 and 2). These investigations conducted on the TCR are intended to improve the understanding of steam-induced stall.

The performance characteristics of the compressor in both the fan-stage as well as the rotor-only configurations were mapped with the data that was collected by Gannon, Hobson and Shreeve. This data was used to establish performance characteristics at 70%, 80%, 90% and 100% design speed prior to and during stall. (Ref 3-4). Unsteady pressure measurements at 60%, 70%, and 80% design speed were reestablished by Rodgers in 2003 (Ref. 5). Inlet and exit surveys at 70%, 80%, 90% and 100% design speed with a three-hole probe were conducted by Villescas (Ref. 6). Villescas determined the spanwise distributions of the rotor diffusion factor at choke, peak efficiency and stall while Brunner repeated the surveys with a 5-hole probe and determined the pitch angle and Mach number distributions in the inlet of the rotor (Ref 7). Payne took performance data at 95 percent speed, with a hot-film probe as well as Kulite pressure transducers in the case wall. He also took transient data from both the hot film and Kulite pressure transducers during steam induced stall at 70% speed (Ref 8).

Changes to the inlet and honeycomb have altered the performance characteristics of the Transonic Compressor Rig. The compressor performance was remapped in order to determine a new stall line as well as the peak performance criterion. In the current study, unsteady pressure measurements were established, with the installation of 9 Kulites at varying positions along the case wall. Data were taken at the 90% design speed as well as prior to and during a steam induced stall. Steam induced stall measurement were also taken at 70% speed to reestablish previously taken data to reflect alterations in the honeycomb. The data was then reduced and analyzed through the use of a data acquisition and data reduction system.

II. TRANSONIC COMPRESSOR

A. SANGER STAGE

The Sanger compressor stage was specifically designed for testing and assessment at the TCR using CFD techniques, while minimizing conventional empirical design methods. Figures 1 and 2 show a sectioned drawing and the rotor installed into the test rig, respectively.

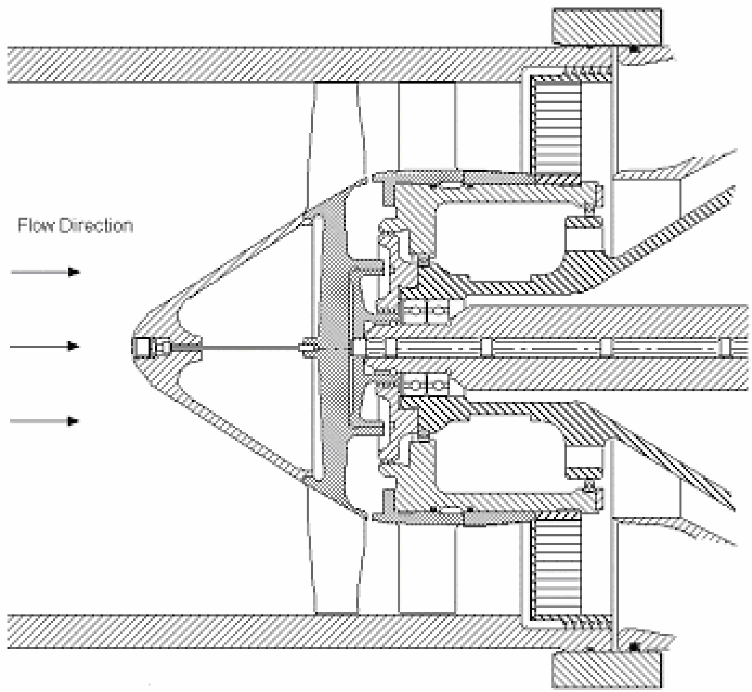

Figure 1. Transonic compressor sectioned view

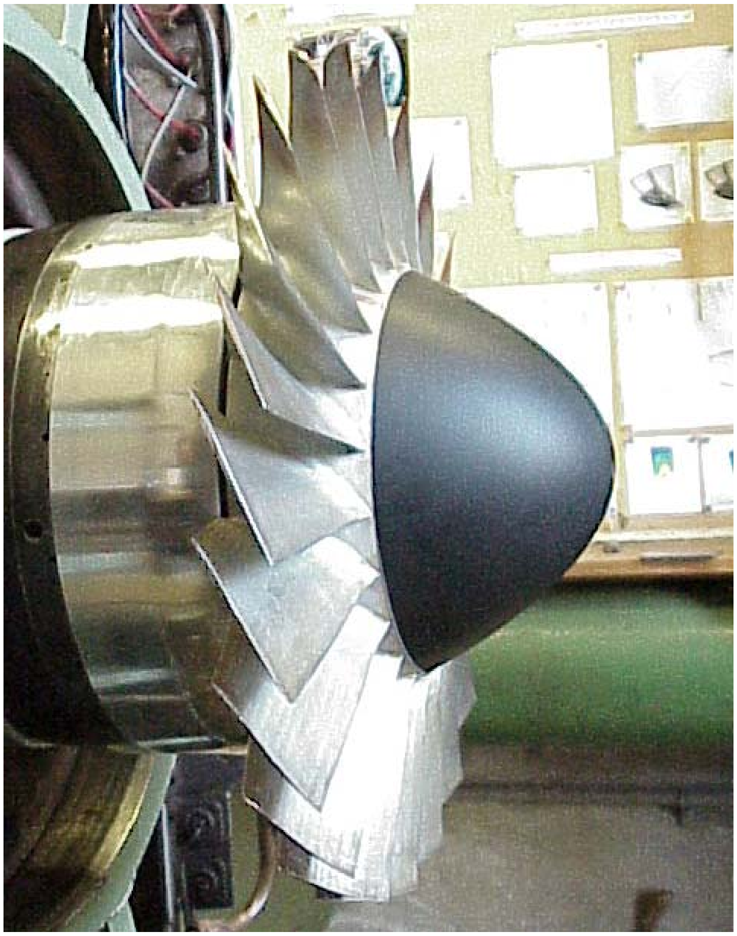

Figure 2. Rotor-only configuration of the transonic compressor rig (From Ref. 8)

The rotor had 22 blades and was made from a high strength aluminum alloy (7075-T6). For the present experiment, the rotor was tested with a parabolic spinner, which replaced the conical spinner used by O'Brien (Ref. 9).

For most of the previous studies the entire stage was evaluated. However, for the current study, the stator was removed and only the rotor was present to ensure the simplest configuration tested during steam ingestion. The design specifications for the Sanger Stage are given in Table 1.

Table 1. Sanger Stage Parameters

Parameter	Value
Pressure Ratio	1.61
Tip Speed	33.02 m/s (1300ft/s)
Design Speed	27085 rpm
Design Mass Flow	7.75 kg/s (17.05lb/s)
Specific Mass Flow	170.88 kg/s-m^2 (35lbm/s-ft^2)
Specific Head Rise	.246
Tip Inlet Relative Mach Number	1.28
Aspect Ratio	1.2
Hub/Tip Radius Ratio	.51
Rotor Inlet Ramp Angle	28.2
Number of Rotor Blades	22
Tip Solidity (Rotor)	1.3
Outside Diameter	0.2794m (11 inches)
Rotor Diffusion Factor – Tip	.4
Rotor Diffusion Factor - Hub	.47

The Sanger stage represents characteristic of a first stage of a modern fan. The tip inlet relative Mach number is lower than most modern transonic compressors, however the blade loading is higher, which allows a pressure ratio of 1.56.

B. TRANSONIC COMPRESSOR TEST RIG

The Transonic Compressor Rig (TCR) test rig, as shown in Figure 3, was driven by two opposed-rotor air turbine stages, supplied by a 12 stage Allis-Chalmers axial compressor. The Allis-Chalmers compressor supplied three atmospheres of air pressure at a mass flow rate of 5 kg/s. Air was drawn into the TCR from the atmosphere through a throttle valve as shown in Figure 4. A five-meter long 46cm diameter pipe connected the settling chamber to the test compressor. The air would then flow through a nozzle, which was used for flow rate measurements and was exhausted back to the atmosphere. The rig's schematic is also shown in Figure 5.

Figure 3. Transonic Compressor in test cell with inlet piping removed (From Ref. 8)

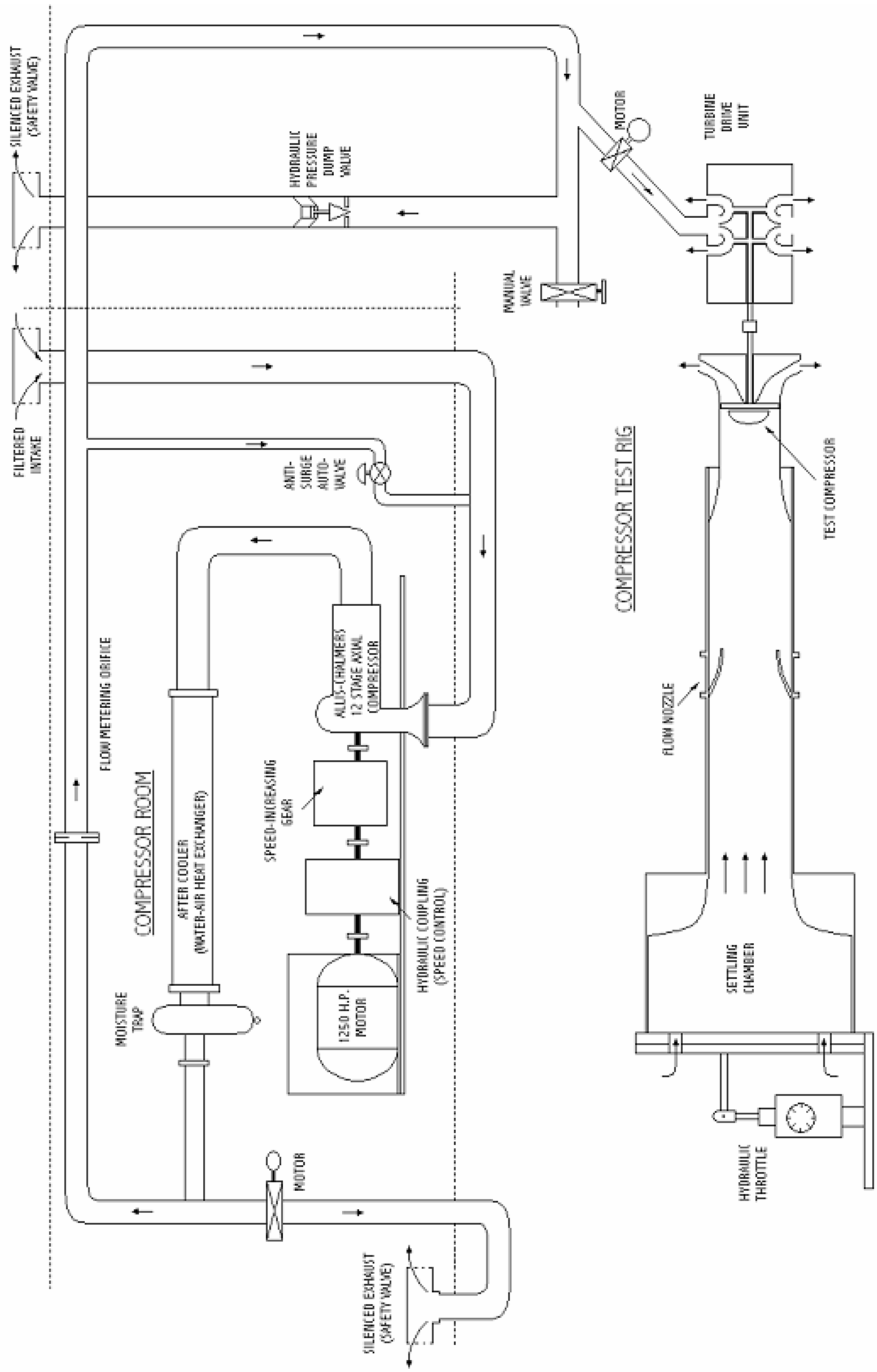

Figure 4. Transonic Compressor Rig Schematic

C. STEAM INJESTION SYSTEM

The compressor rig and steam ingestion system is shown in Figure 5. Steam was generated by the SVS600 steam boiler and was directed through a 7.62 cm diameter pipe and vented to the intake plenum as can be seen in Figure 6. The SVS600 was capable of producing saturated steam up to a maximum working pressure of 1000 kPa or 1.4 kg/sec at 100 ^{0}C and can be seen in Figure 7 (Ref. 9). In order to monitor the transient response of the steam pressure, a pressure transducer was installed into the steam pipe and can be seen in Figure 5. Two remotely-operated fast acting solenoid valves were used for releasing the steam into the intake plenum as well as for venting the pipe. Figure 7 shows the orientation of the steam pipe with respect to the intake plenum.

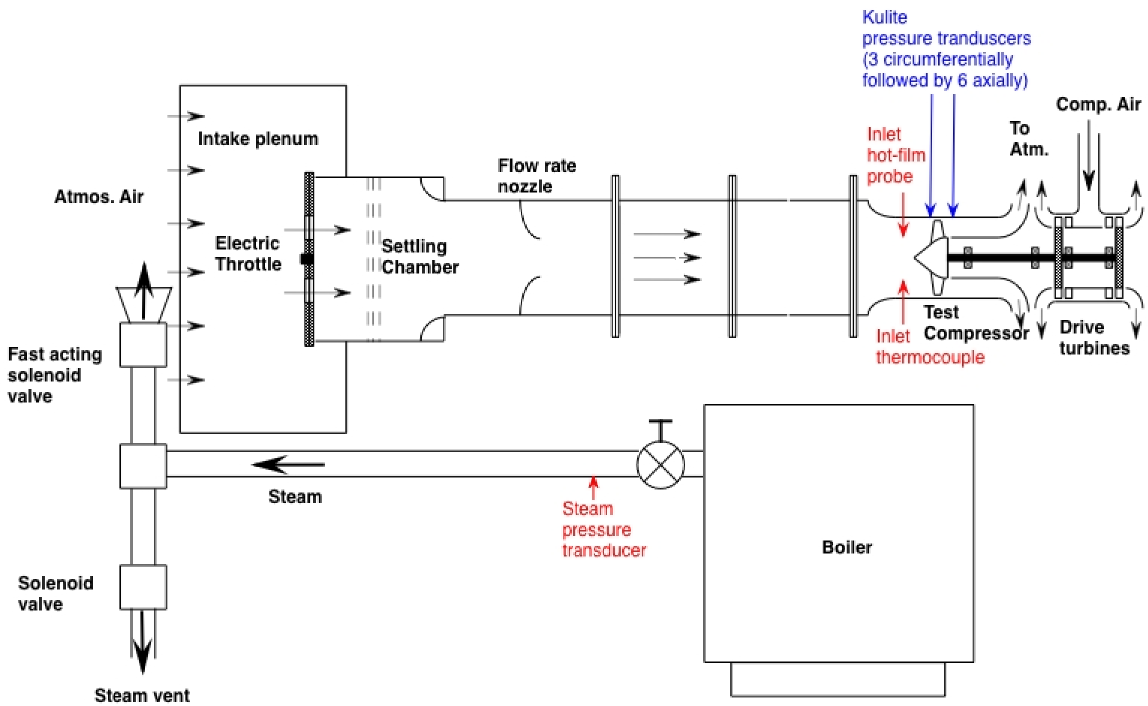

Figure 5. Transonic compressor rig with steam ingestion system

Figure 6. SVS600 steam boiler system

Figure 7. Steam pipe with intake plenum orientation (From Ref. 8)

THIS PAGE INTENTIONALLY LEFT BLANK

III. INSTRUMENTATION

A. KULITE PRESSURE TRANSDUCER

A Kulite miniature silicon pressure transducer, model XCQ-080-25, was used to obtain time-resolved pressure data. The probe was a miniature, semiconductor, strain gauge transducer which incorporated a fully active four-arm Wheatstone bridge dielectrically isolated silicon-on-silicon diaphragm. A diagram of the probe is given in Figure 8 and the specifications are given in Table 2.

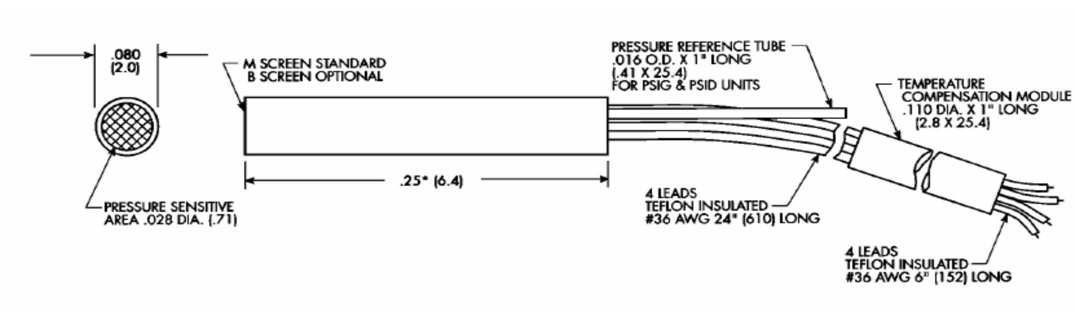

Figure 8. Kulite XCQ-080 series transducer (From Ref. 5)

Table 2. XCQ-080-25 Factory Specifications (From Ref. 5)

Input		
Pressure Range	1.7 atm	25 PSI
Over Pressure	3.4 atm	50 Psi
Burst	5.1 atm	75 Psi
Rated Electrical Excitation	10	VDC/AC
Maximum Electrical Excitation	15	VDC/AC
Input Impedance	800	Ohms
Output		
Output Impedance	1000	Ohms
Full Scale Output	100	mV
Residual Unbalance	+-3	% FSO
Non-Linearity and Hysterisis	0.1	% FS BFSL
Hysteresis	0.1	%
Repeatability	0.1	%
Resolution	Infinite	
Natural Frequency	300	kHz
Perpendicular Accel Sensitivity	0.0003	% FS/g
Transverse Accel Sensitivity	0.00004	% FS/g
Insulation Resistance	100	Megohm
Environmental		
Operating Temp Range	-53.8 to 121.1 deg C	(-65 to 250 deg F)
Compensated Temp Range	26.7 to 82.2 deg C	(80 to 180 deg F)
Thermal Zero Shift	+- 1	% FS/100 F
Thermal Sensitivity Shift	+- 1	% FS/100 F

For the current study, 9 Kulite transducers had to be installed in the casing of the TCR. The full bridge Kulite Pressure Transducers were connected to the Hewlett-Packard E1529A Remote Strain Conditioning Unit, via a RJ-45 cable (Ref. 10). Figure 19 shows the correct set up of the Kulite wires and their corresponding RJ-45 pin assignments.

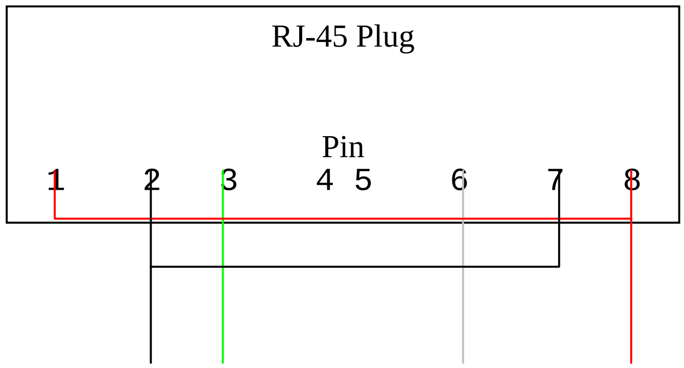

Figure 9. Kulite Connection to RJ-45 Plug (From Ref. 5)

The Kulite Pressure Transducer had four wires, a black, white, green, and red wire. The black wire was connected to pins 2 and 7, and the white was connected to pin 6. Pin 3 was connected to the green wire and pins 1 and 8 were connected to the red wire. Pins 4 and 5 were not connected to any wires.

B. INSTALLATION OF KULITE PRESSURE TRANSDUCER

Nine Kulites were connected to RJ-45 cables and were installed into aluminum slugs. The aluminum slug was originally designed by Vavra for unsteady pressure measurements of the Vavra stage, and can be seen in Figure 10 (Ref. 11).

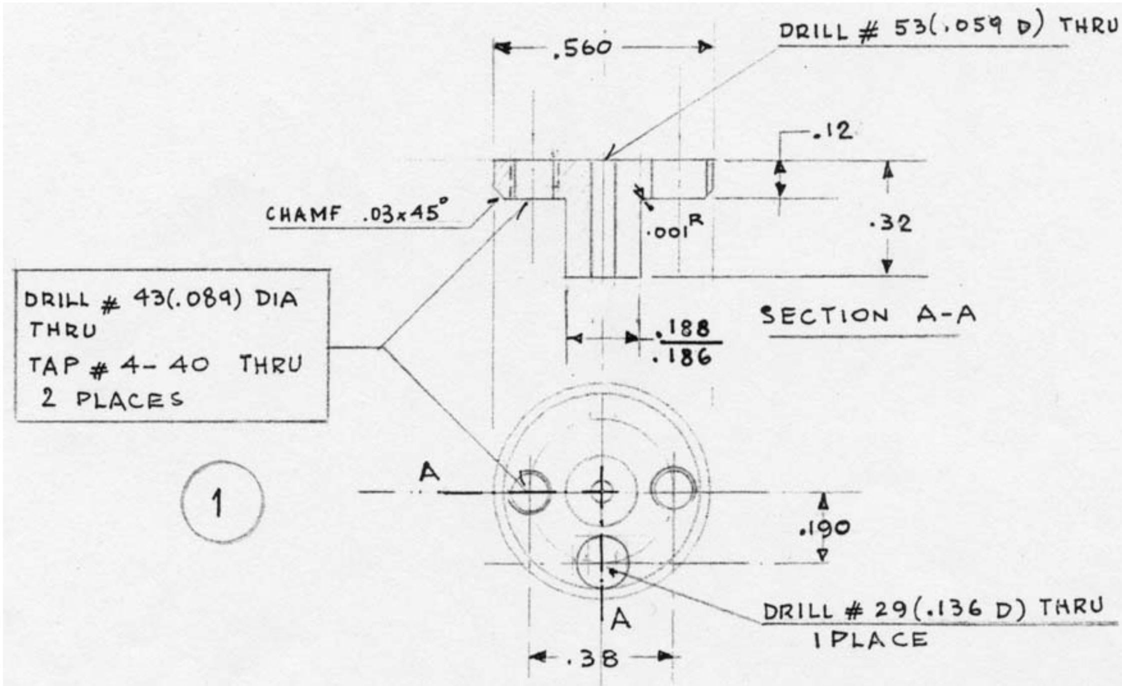

Figure 10. Kulite Mounting Design

Once the nine Kulite pressure transducers were installed into the aluminum slugs they were mounted flush with the casewall. Figures 11 and 12 show the positioning of the Kulites relative to the casewall. The Kulite pressure tap locations were spaced one blade spacing apart and their corresponding locations across the blade were at 10.5%, 37%, 63%, and 89.5% axial chord. In addition a once-per-revolution speed pickup was used.

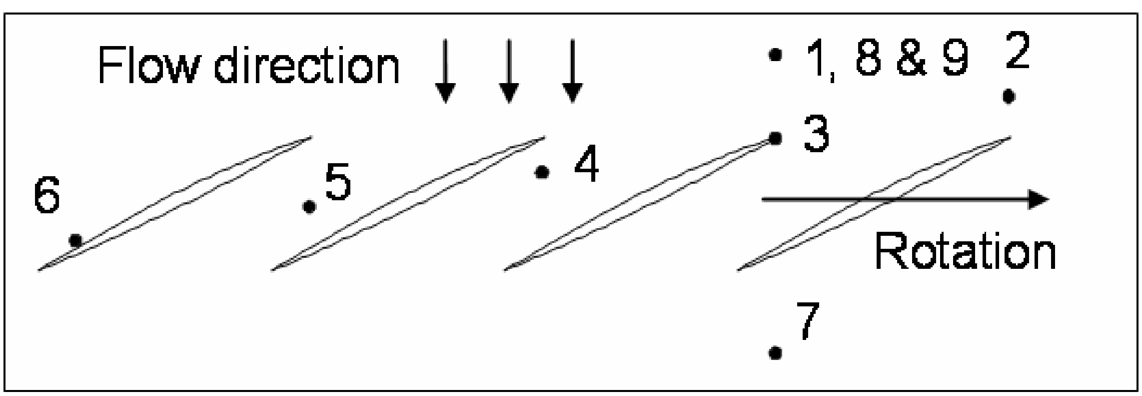

Figure 11. Relative positions of Kulite pressure transducer and blades. (From Ref. 5)

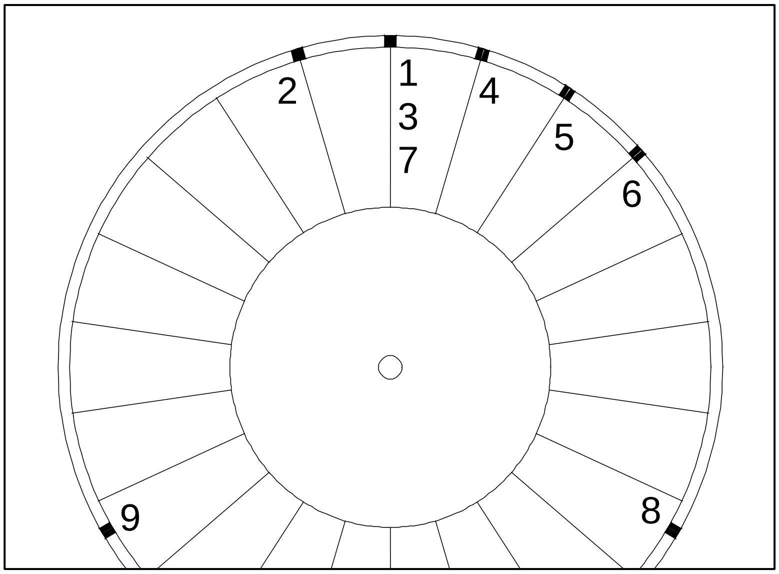

Figure 12. Relative positions or Kulite pressure transducers in case wall (From Ref. 5)

C. DATA ACQUISITION

The Kulite pressure transducers were connected to a Hewlett-Packard E1529A Remote Strain Conditioning Unit via an RJ-45 LAN cable. An adjustable power supply would provide the necessary excitation voltage of 5 Volts and was input into the bridge excitation port of the HP E1529A. An Remote Channel Multi-Function DAC Module (HP E1422A), (Ref. 12), controlled and set the HP E1529A to a full bridge configuration, calibration, and self test functions via a program written in HP Vee Pro, (Ref. 13). The HP E1529A provided a wideband amplified output from each strain bridge signal, via a 37-pin connector, to a HP E1433A high-speed digitizer, capable of taking samples up to 196 kSa/sec (Ref. 14). A tachometer signal was also connected to the HP E1433A to provide speed reference data. The tachometer signal came in via standard coax cables and a break-out box was used as an adapter to route these signals into the 27 pin connection of the HP E1433A. The HP E1433A and HP E1422A were addressed through the HP E8404A VXI Mainframe and interfaced to a PC. The data was stored on an Agilent N2216A VXI/SCSI Interface Module, containing two internal 50 Gbyte drives (Ref. 15). The VXI Mainframe was interfaced to a PC, with a 'firewire' interface. DAC

Express, an Agilent program, was used to acquire data. Figure 13 shows the connection of the Kulite transducers to the data acquisition system.

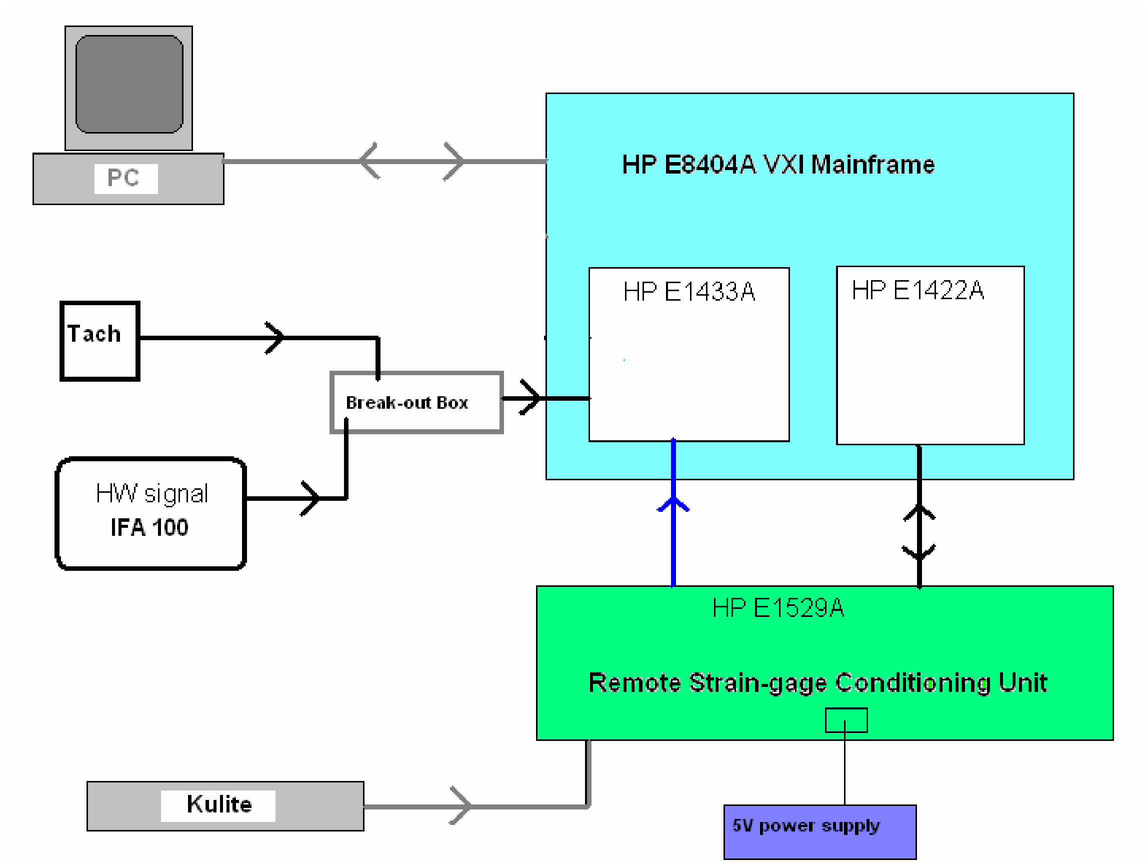

Figure 13. Data Acquisition System (From Ref. 5)

1. DAC Express

The Hewlett-Packard DAC Express was used to monitor the digitized signal in real time (Ref. 16). DAC Express set the sampling rate of the HP E1433A and recorded the digitized data to the Agilent N2216A. DAC Express can analyze up to 16 channels at a time, with the option of recording for any given amount of time. Figure 14 shows an example DAC Express setup screen showing 12 channels in real-time, nine individual Kulites, Kulites 2,7 and 8, the hotwire and the once per rev signal. For the purposes of this experiment, the length of time for the transient pre-stall was .2 seconds, while the two stall cases were 45 seconds. However, for ease of data analysis the stall cases were later reduced to four seconds; two seconds before stall and two seconds after stall. Once

the start button had been pushed and the data recorded, the data had to be exported from the N2216A to the PC as a .csv file. The .csv files were entered into MATLAB, for data processing (Ref. 17).

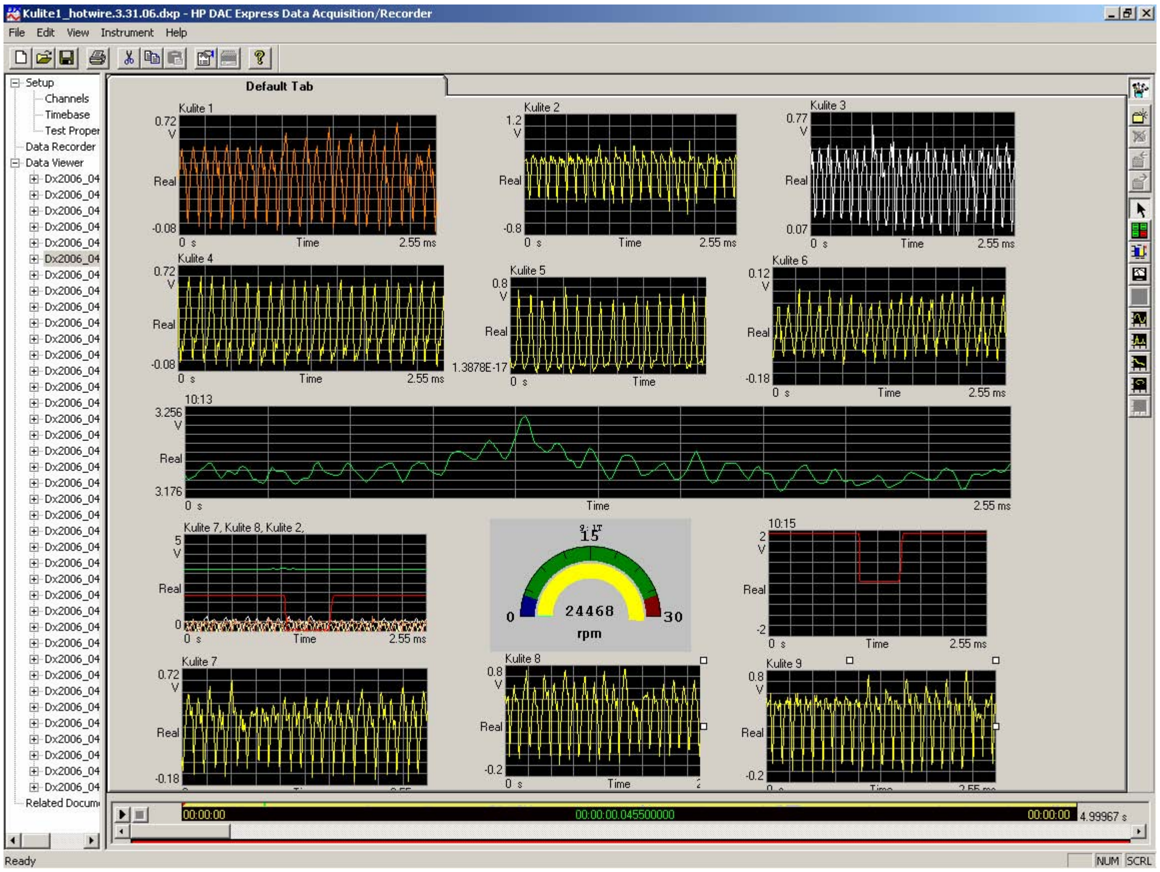

Figure 14. DAC Express GUI screen

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EXPERIMENTAL PROCEDURE

A. KULITE CALIBRATION

Calibration of the Kulite was conducted while the compressor was running, to alleviate any temperature dependence of the Kulite. Four sets of data were taken for the specified throttle and speed setting and each set of data was applied to a reference pressure. The reference pressures were 0, 5, 10 and 15 inches of mercury (0, 2.456, 4.912, 7.368 psig). The applied reference pressure was manually recorded from a Wallace and Tiernan gauge with a mirrored scale graduated in .2 inches of mercury. For the steady state analysis, the DAC Express would record the voltages for .2 seconds, at all four reference pressures. However, for the two stall cases only the reference pressure of 10 psig was recorded into the DAC Express for 45 seconds. Once the data had been recorded as a .csv file it was calibrated and reduced in MATLAB (Ref. 17). The calibration constants calculated in MATLAB from the run closest to stall were used for data analysis and data reduction of the stalled data.

B. COMPRESSOR OPERATION

During testing the Transonic Compressor Rig's rotor was kept at a constant speed and measurements were taken at different throttle settings. By closing the throttle, the mass flow rate was reduced and the rig operating point could be determined by the procedures described by Gannon et al. (Ref. 4). For this experiment the mass flow rate was varied by actuating the throttle (Fig. 15) while the rotor RPM was set at a particular speed.

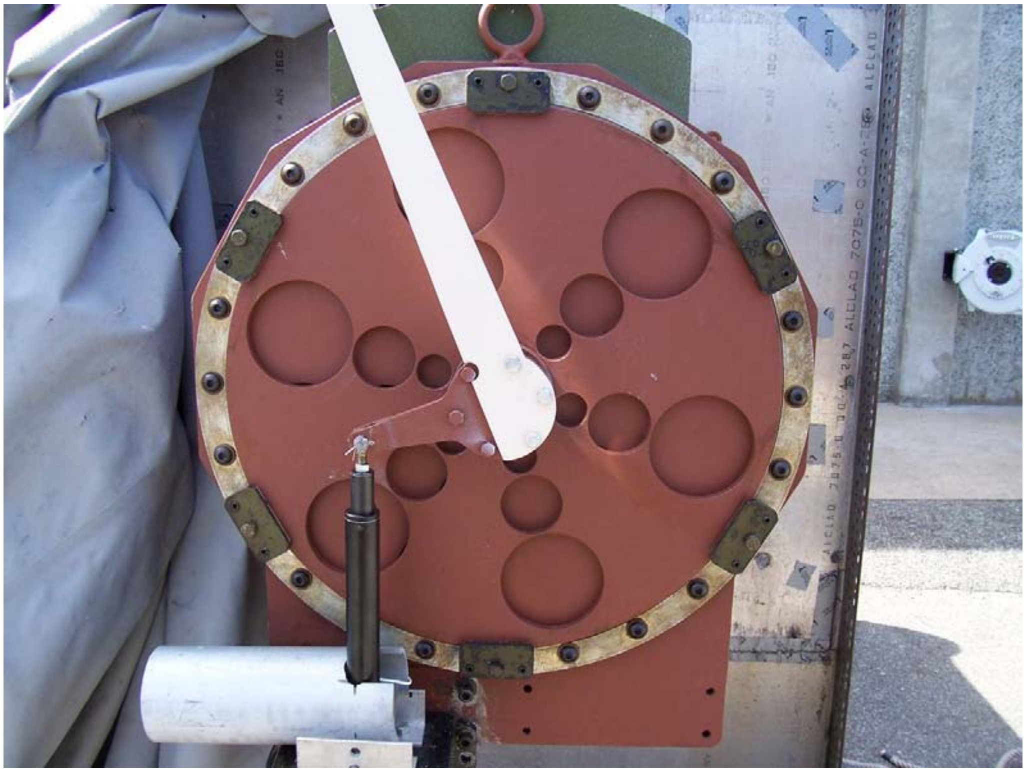

Figure 15. Electric throttle

Mass flow rate, inlet and exit total temperatures and pressures were measured and recorded. This data was used to calculate total-to-total pressure ratio and isentropic efficiency which was used to determine position on the compressor map. Measurements were taken from open throttle to a throttle position near stall. Measurement and calibration procedures were described in more detail by Gannon et al. (Ref. 3).

C. STEAM-INDUCED STALL RUNS

For the current study the compressor was set at either 70% or 90% speed. The throttle was closed incrementally, pausing only to take the necessary steady state measurements. The process of reducing the mass flow by closing the throttle was done to determine the point at which the rotor would stall before moving on to steam induced stall test.

Once the throttle setting just prior to stall was established, and steady state measurements were taken, the compressor was ingested with steam by the following

procedure. The steam vent solenoid valve and the boiler isolation valve were opened to allow the steam pipe to heat up. Once fully heated, the vent valve was closed and the data trace from the pressure transducer and thermocouple were started. The steam pressure was monitored and once it reached its intended pressure, the isolation valve was closed. At this point a three second countdown to steam ingestion would occur and the Kulite data acquisition system was initiated. Once the end of the countdown had been reached, the fast-acting solenoid valve was opened and the steam was dumped into the plenum of the compressor. After several seconds, the Kulite data acquisition was stopped. If the steam ingestion did not cause a “pop-stall” event, the procedure was repeated at a reduced throttle setting until a “pop-stall” was achieved.

Post processing of the data was conducted with MATLAB (Ref. 17). The procedure for the use of the MATLAB files is presented in Appendix A. Specific M-files that were used with the steady-state data are given in Appendix B and the procedure for the stall tests are presented in Appendix C.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS AND DISCUSSION

A. STEAM-INDUCED STALL AT 70% SPEED

Given that changes to the honeycomb and pressure ports on the flow rate nozzle were altered the performance characteristics of the Transonic Compressor Rig, transient data using Kulite pressure transducers during steam induced stall at 70% speed needed to be re-established. Figure 16 shows previous data taken on the rotor with a pneumatic temperature and pressure probes, torque, flow, and speed instrumentation, and represents time-averaged information. Figure 16 also shows the single point near stall at 70% speed that was measured prior to steam ingestion (Ref. 18). The green dot was established by Payne and the blue dot demonstrates shift along the stall line due to the changes in the honeycomb and pressure ports (Ref. 8).

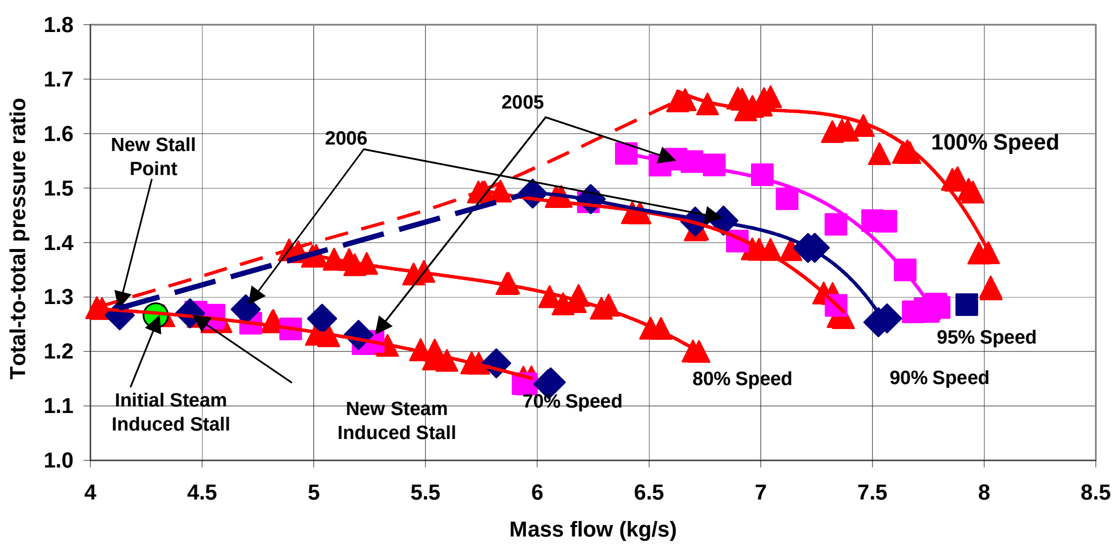

Figure 16. Pressure ratio versus mass flow rate with shift in stall line

The transient pressure measured in the steam line was used to calculate the mass flow rate of the ingested steam. Figure 17 shows the pressure change in the steam pipe as well as the temperature change in the inlet of the compressor over time during the steam-induced stall experiment. A pressure of 480 kPa was reached in the steam pipe prior to

releasing steam into the compressor. A steam-induced stall of the rotor was observed at 70 percent speed at a mass flow rate of 0.045 kg/s.

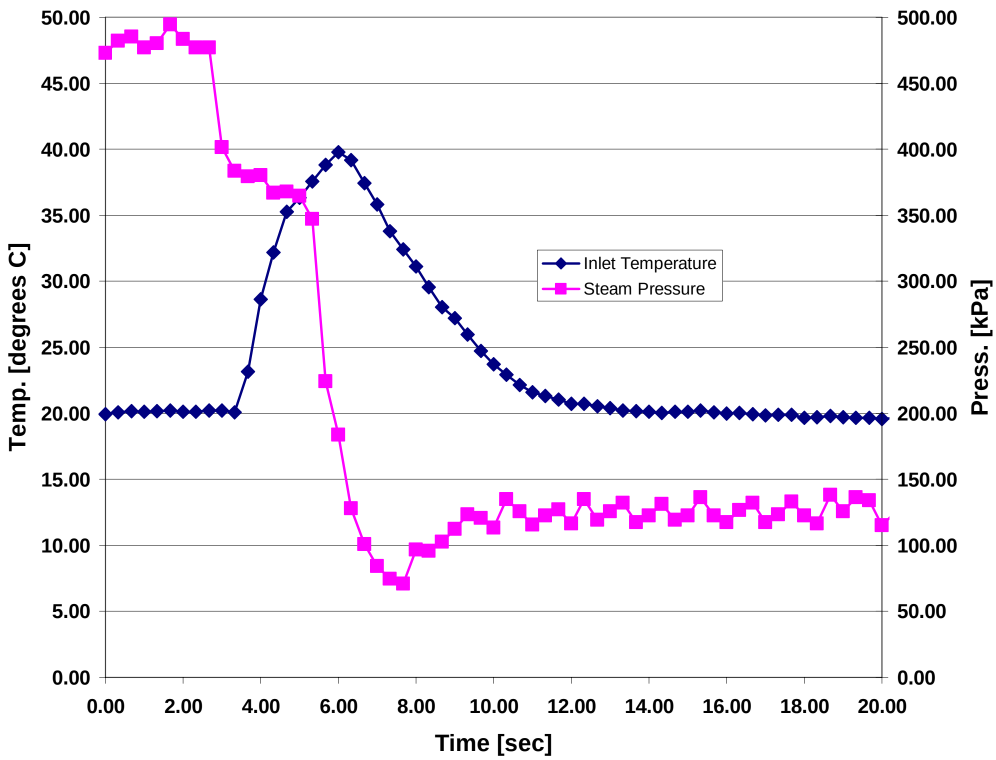

Figure 17. Steam pressure and inlet temperature change during steam-induced stall at 70 percent speed

Figure 18 shows the raw-voltage of the Kulite signal going into stall at 70 percent speed. The change in speed of the compressor going into stall is shown in Figure 19.

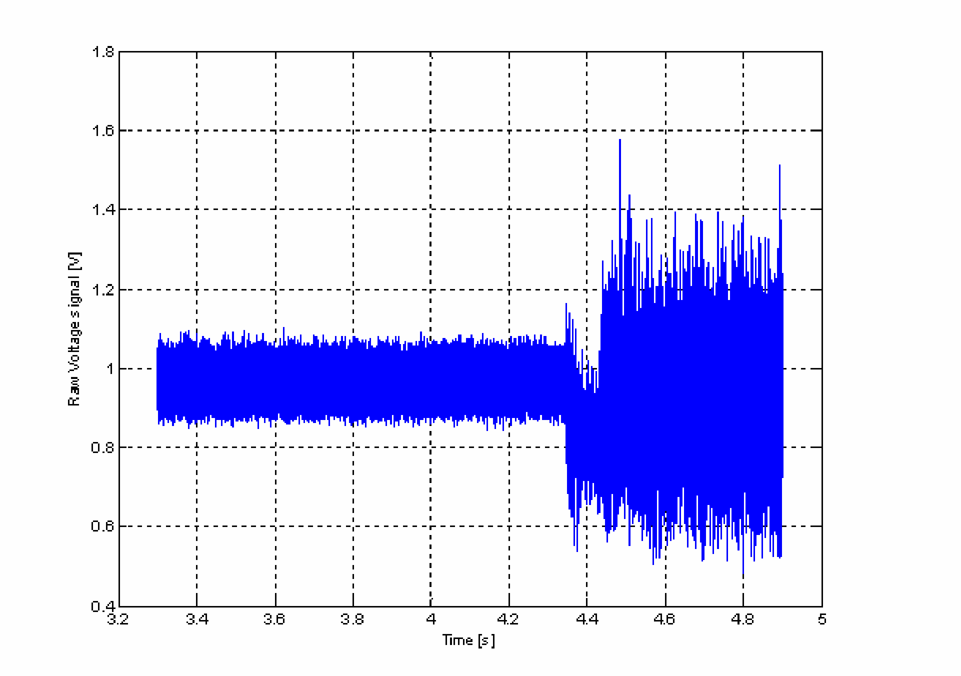

Figure 18. Raw-voltage Kulite signal going into steam induced stall at 70 percent speed

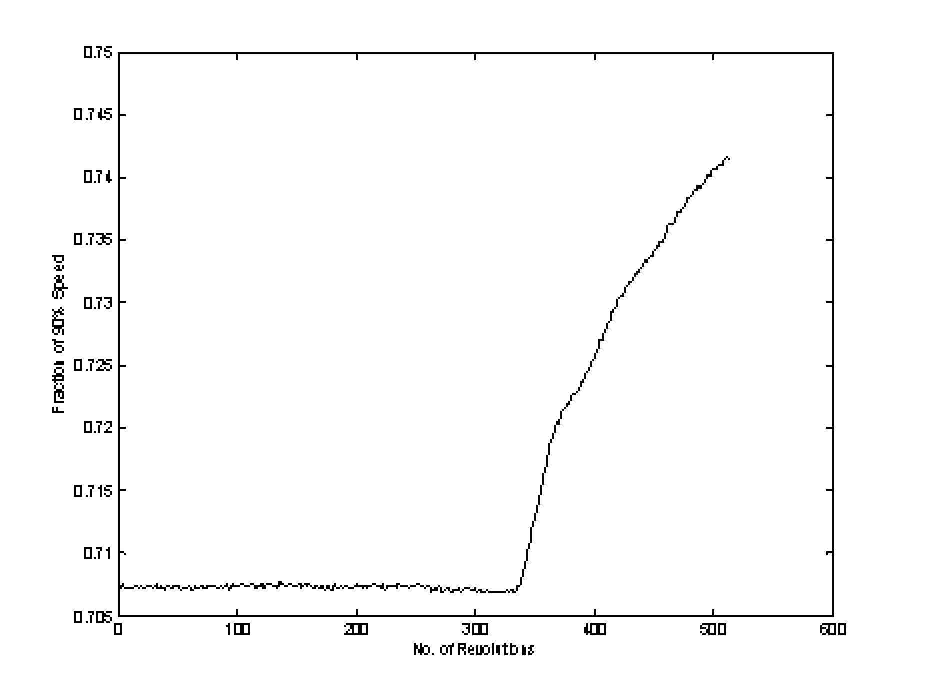

Figure 19. Change in compressor speed during steam-induced stall

Figure 20 represents a waterfall FFT contour plot of the data taken during stall at 70 percent speed. The stall cell frequency, once per revolution, and the blade-passing frequency, can be seen in this contour plot. The stall cell frequency was approximately 60 percent of rotor speed and there was indication of a precursor to stall at 4.3 seconds. Stall occurred at 4.34 seconds.

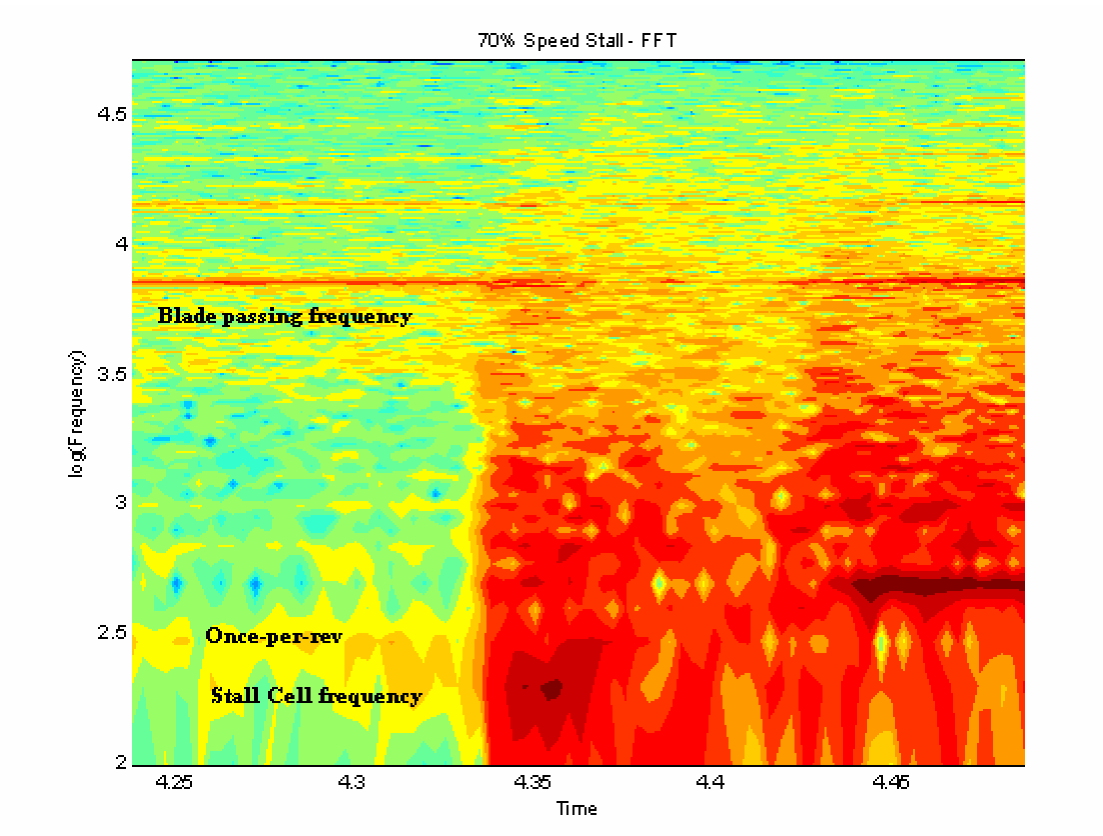

Figure 20. Power spectrum contour plot of Kulite data at 70 percent speed through stall

B. PRESSURE CONTOURS 70% SPEED OPEN THROTTLE

Figure 21 shows the pressure contours derived from the nine Kulite transducers at 70% speed.

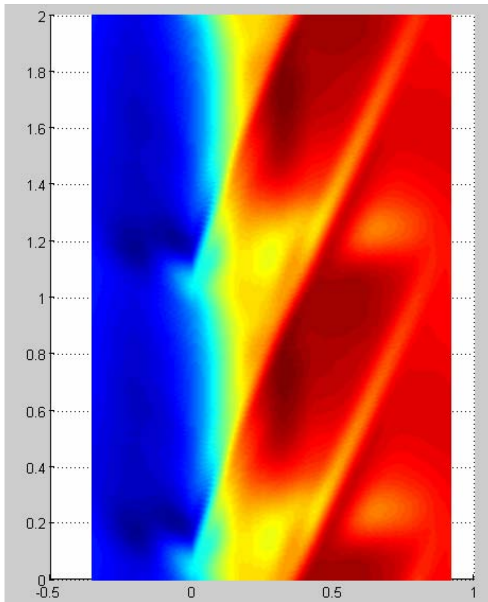

Figure 21. Pressure contours 70% speed with open throttle

C. STEADY-STATE PRESSURE CONTOURS AT 90% SPEED

The process of reducing the mass flow by closing the throttle while keeping the compressor at a constant speed of 24375 RPM, was done to determine the point at which stall would occur. During this process, the reduction of the mass flow rate resulted in a reduction of axial velocity into the fan and for constant rotational speed, this yielded an increase in incidence. This increase in incidence increased the force on the blades, which yielded an increase in static pressure rise (Ref. 19). Table 3 shows the pertinent measurements while closing the throttle. Run 4 does not include efficiency, or ΔP measurements because this was the run that was closest to stall and steady state data was not able to be taken.

Figures 22-25 are the graphical pressure contours of Table 3. Note the shock wave which moved forward as the mass flow is decreased.

Run	Efficiency (η)	Mass flow	ΔP	Pressure ratio
1	86.55	7.56	6.54	1.25
2	89.85	7.23	5.25	1.38
3	83.12	6.24	3.76	1.47
4	-	5.98	-	1.49

Table 3. Measurements during transient analysis

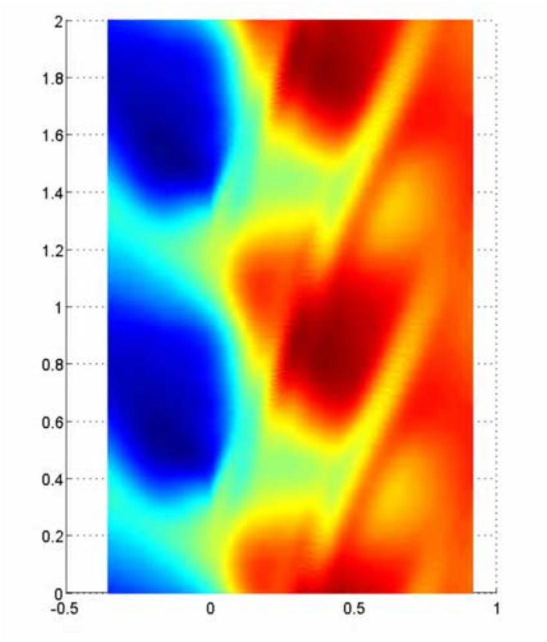

Figure 22. Pressure Contours 90% speed and a Pressure Ratio of 1.25 (Open throttle)

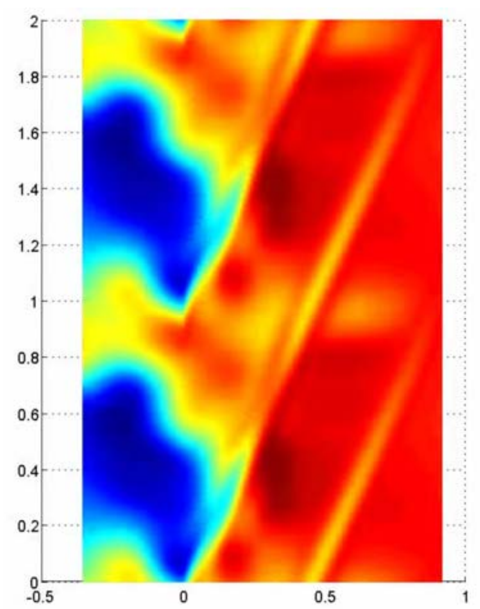

Figure 23. Pressure Contours 90% speed and a Pressure Ratio of 1.38 (Peak Efficiency)

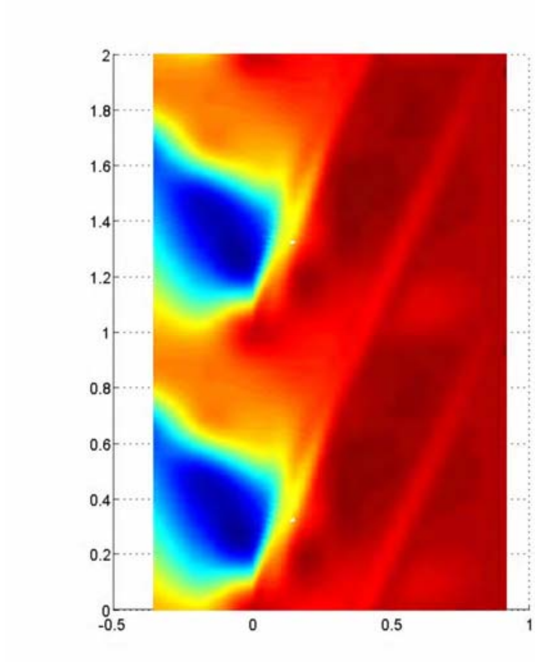

Figure 24. Pressure Contours 90% speed and a Pressure Ratio of 1.47

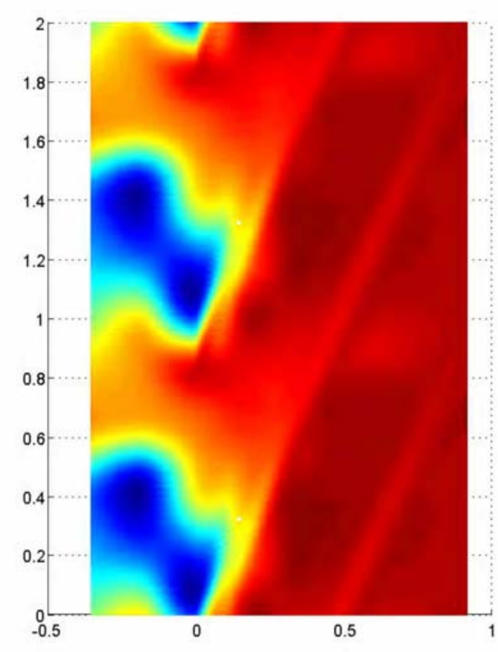

Figure 25. Pressure Contours 90% speed and a Pressure Ratio of 1.49 (closest to stall)

B. TRANSIENT MEASUREMENTS DURING AT 90% SPEED

In order to gather data during a stall event, the compressor was run as close to stall as possible. Once the high speed data acquisition system was activated, the upstream throttle was closed until stall occured. As soon as stall occurred the upstream throttle was reopened and stall stopped as soon as possible to reduce the adverse loading on the compressor. Figure 26 shows the raw-voltage of the Kulite signal going into stall at 90 percent speed.

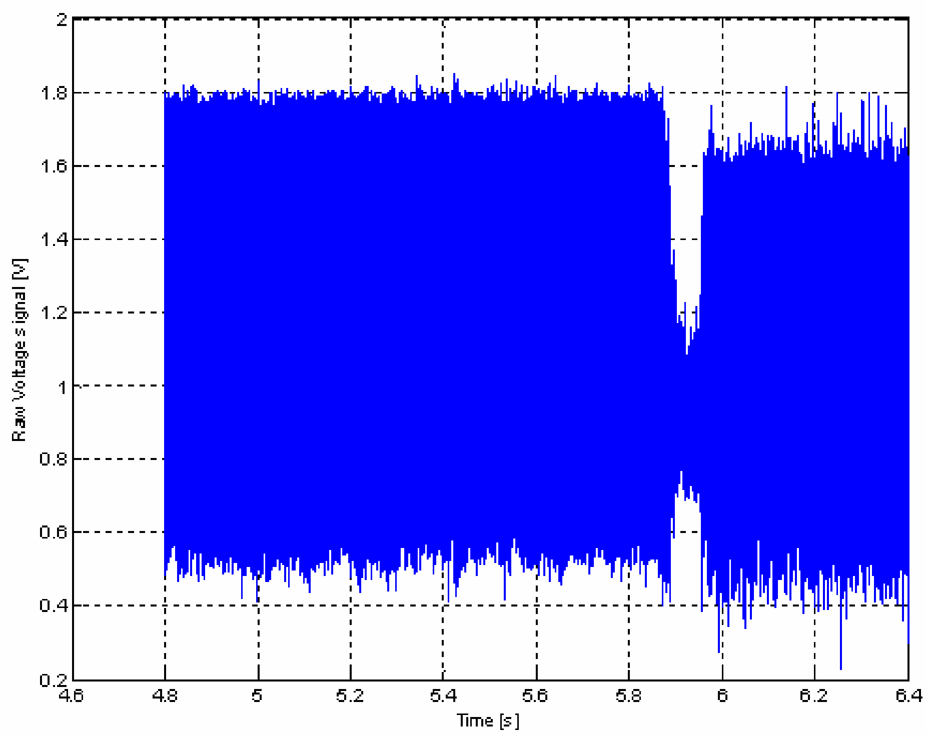

Figure 26. Raw Kulite data through 90% speed stall

In order to post-process the data the stall speed cell needed to be determined. This speed was acquired through the use of three pressure probes at the same axial location but just upstream of the blades. Figure 27 shows the simultaneous raw voltages at 90% speed.

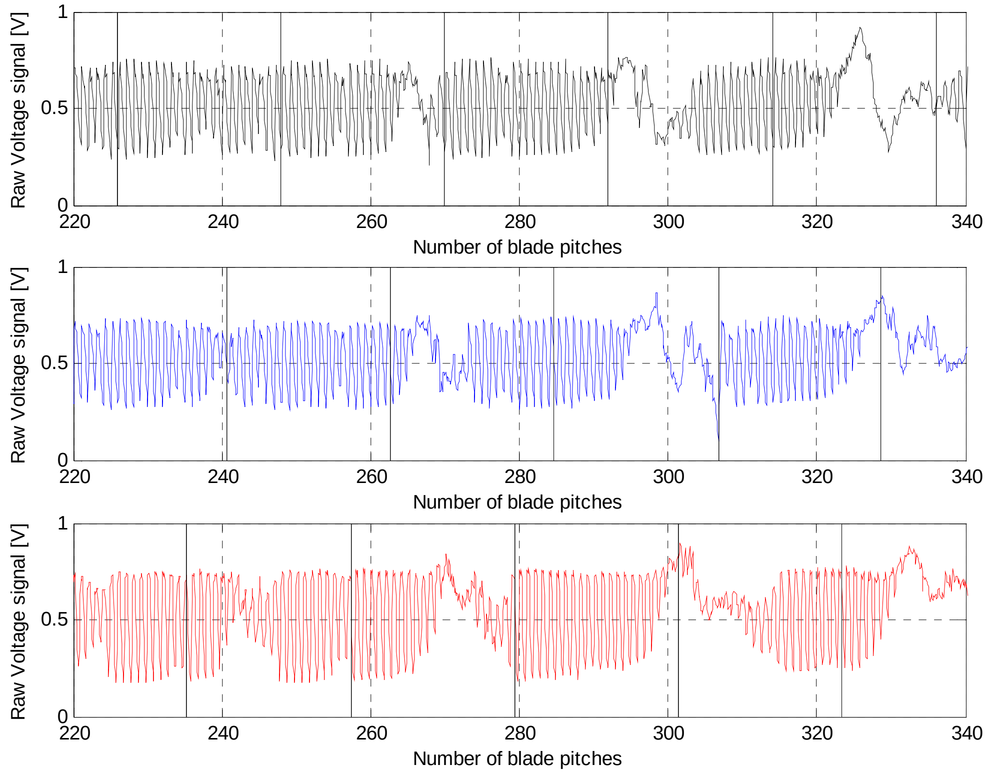

Figure 27. Simultaneous signal obtained from 3 probes at 90% speed

The stall cell size increased with each revolution but was separated by a region of regular cyclic flow. A full investigation into this phenomenon at different speeds is given by Gannon et. al. (Ref. 20). The change in speed of the compressor going into stall is shown in Figure 28. A fast Fourier transform was also used to create a waterfall power spectrum of the Kulite data as seen in Figure 29 plotted in contour format.

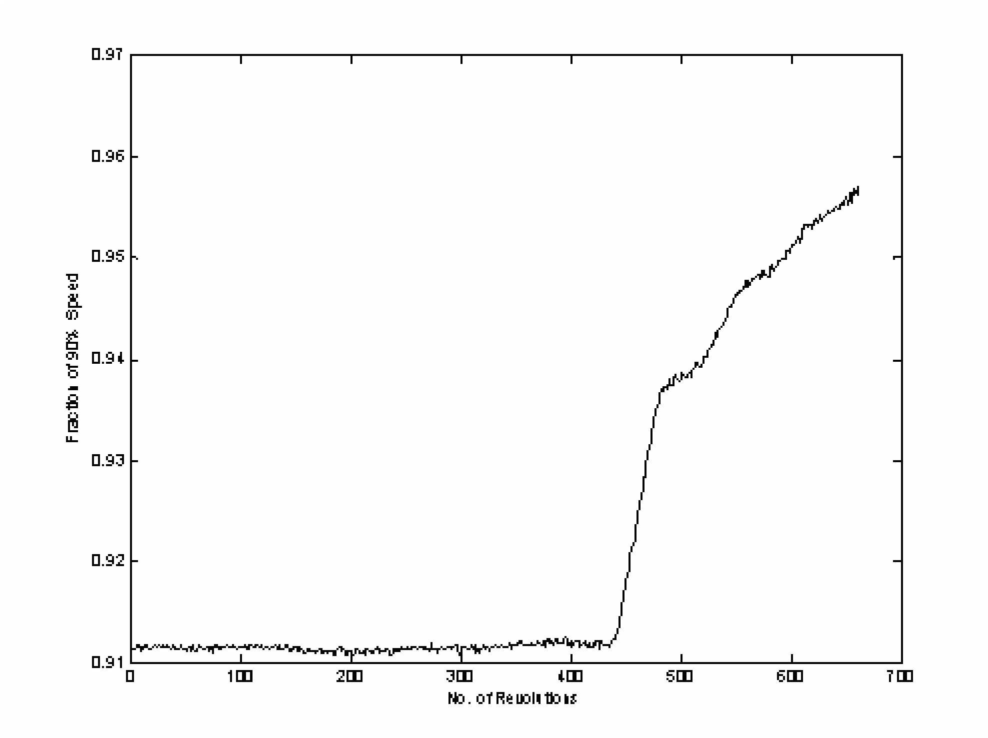

Figure 28. Change in compressor speed during a rotating stall event

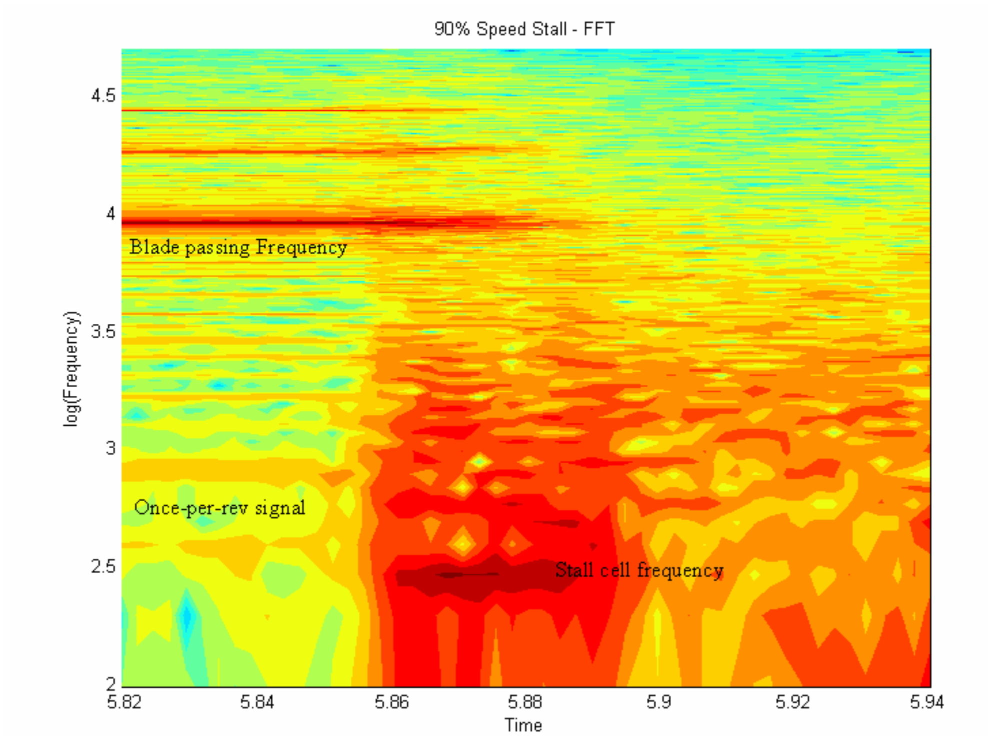

Figure 29. Power spectrum contour plot of Kulite data at 90 percent speed at through stall

Figure 29 represents data taken during stall at 90 percent speed. The stall cell frequency, once per revolution, and the blade-passing frequency, can be seen in this contour plot. There was no indication of a stall precursor as measured by the Kulite transducers.

C. 90% SPEED STALL CELL GROWTH

Figure 30 represents the development of the stall cell at 90% speed. The first strip corresponds to the undistributed rotation before the formation of the stall cell. Each subsequent strip corresponds to the stall cell as it passed under the pressure probes on the subsequent rotation. Figure 30 shows the formation of only one stall cell rotating at 60% rotor speed,

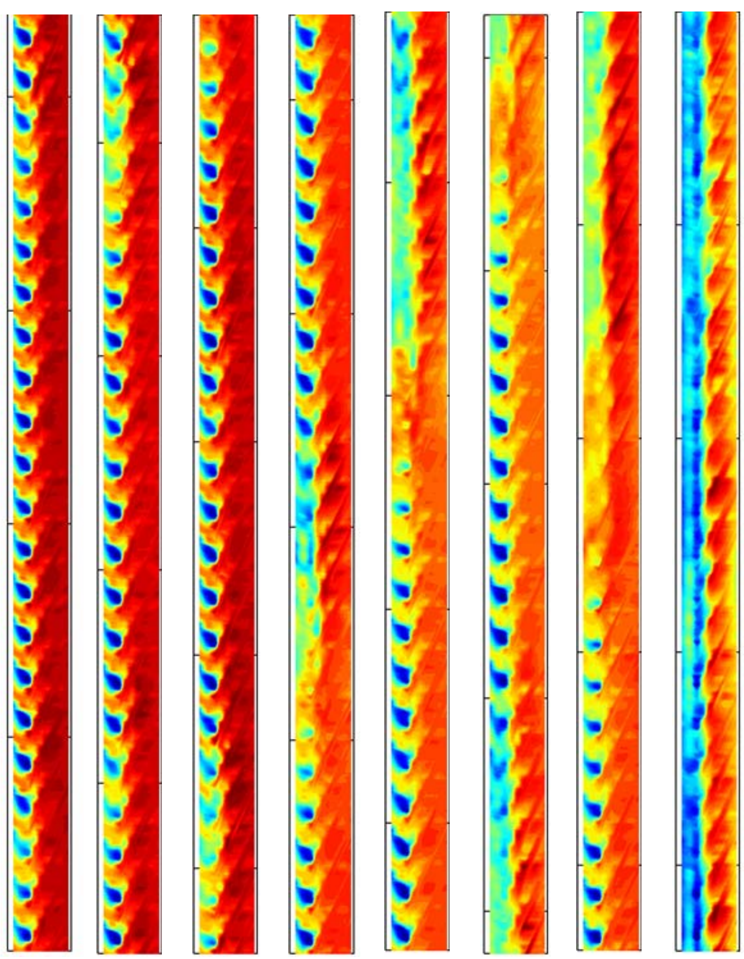

Figure 30. 90% speed stall cell growth

Once the stage was stalled it needed to be returned to the original flow conditions. Note that because the speed of the machine was not held at a constant speed during the stall event, the un-stalling of the stage was much less controlled than the entry into stall. As covered by Gannon, when the required pressure ratio across the stage decreases the flow through the rotor, the flow begins to return to its axi-symmetric pattern (Ref. 20).

VI. CONCLUSION

Changes in the honeycomb altered the performance characteristics of the Transonic Compressor Rig. As a result, a steam induced stall at 70 percent speed and new stall line were successfully determined. The transient pressure was measured in the steam line and the mass flow of the steam was established. Kulite pressure transducers recorded the pressures prior to and during steam induced stall. The stall cell induced by the steam rotated at 60 percent of rotor speed with slight precursor for 0.04 seconds.

Performance measurements were carried out at 90 percent speed from open throttle to stall. This data followed previous trends with respect to peak efficiency and total pressure ratio. A graphical analysis of the data was used to better visualize the structure of the flow as well as the stall cell and its growth. The measured stall cell also rotated at 60 percent rotor speed with little or no indication of any precursor.

A steam induced stall at 90 percent speed is planned. With the aid of the graphical techniques presented, insight could be gained into the formation and propagation of the steam induced stall.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: PROCEDURE FOR USE OF MATLAB M-FILES

Stall case

Directory that contains all the files:

C:\Bill_Levis\Kulite_Rotor_Only\90%\90%2004_09_14_Rot_only_0.00%\Stall

load_data_90 is the file which calls all of the others

On line 14 in load_data_90 reads:

%Raw_data = dlmread('Dx2004_0914_1001_90_stall.csv',',','A187500..M200000'); %

'Dx2004_0914_1001_90_stall.csv' is the csv file will be read and needs to be altered to accommodate a new set of data

A187500..M200000 corresponds to the cells and time of the stall event

The timed can be determined by trial and error or can be determined by using the DAC Express to find the exact time of the stall event. And given that each cell corresponds to .00001 seconds one can determine which cells are at the inception of stall. For example, the exact time that was determined using the DAC Express was 5.861 seconds. Given that the data starts at 3.5 seconds. The difference between the two is subtracted and than divided by .00001.

***Note**
Because there is no calibration procedure for the stall case the calibration constants closets to stall will be used in the stall case. Run the steady state programs for the case closest to stall. The calibration constants will be created in:

C:\Bill_Levis\Kulite_Rotor_Only

under the file name “Kulite_calibrate”

for the 90% speed case rename Kulite_calibrate to Kulite_calibrate_90

and place the newly renamed file into

C:\Bill_Levis\Kulite_Rotor_Only\90%\90%2004_09_14_Rot_only_0.00%\Stall directory

load_data_90 can now be run

***Note**

If color scheme isn't working the contour map needs to be reset to within tolerance. This can be accomplished in load_data_90 line:

```
contourf(PR_X,PR_Y,PR_Z,[0.4752:(1.4619-0.4752)/35:1.4619])
```

steady state

There should be 4 csv files for dictated percent speed as well as mass flow corresponding to the different back pressures. As default they need to be saved in this directory:

C:\Bill_Levis\Kulite_Rotor_Only\90%\90%2004_09_14_Rot_only_0.00%\Run11

***Note:** this directory can be changed but alterations to the code need to be made

The following directory contains the necessary files to run the steady state calculations:

C:\Bill_Levis\Kulite_Rotor_Only

kulite_rotor_only is the file which calls all the others

In kulite_rotor_only line 12 needs to be changed:

Kulite_constants = 'Kulite_constants_90_PR_1_49'; % 90% speed near stall

Kulite_constants_90_PR_1_49.m dictates which directory will be used

Again unless one wants to alter the code the following directory needs to be used:

C:\Bill_Levis\Kulite_Rotor_Only\90%\90%2004_09_14_Rot_only_0.00%\Run11

APPENDIX B: MATLAB M-FILES (STEADY STATE)

Kulite_constants_90_PR_1_49.m

```
M function file to store all the constants required

function
[Kulite_Subdir,Kulite_Run_no,Run_nos,Blade_no,Kul_no,colours,Avg_size,Kul_offset,Kul_ax_c
ho,Blade_th,Rho_Hg,g,gam_gas,Diameter,Chord,...
        pitch_plot_n,pitch_time_n,pitch_tang_n,Kul_order,Ps_chan] =
Kulite_constants();

% Put this section in a gui to make it simpler for someone else to use
% The names of the subdirectory to be analysed and the run numbers involved

Kulite_Subdir = strvcat('90%\90%2004_09_14_Rot_only_0.00%'); % Grouping of Run

Kulite_Run_no = strvcat('Run11');                    % Individual Run

Run_nos       = [11 12];                             % Run numbers from notes
text file

% Constants
Kul_no     = [1 2 3 4 5 6 7];                       % Kulite Channel plot order

% In one of the tests the Kulites were mixed up so this step was introduced in
case it happens again
Kul_order  = [1 9 2 3 4 5 6];

colours    = ['b' 'g' 'r' 'c' 'm' 'k' 'y'];          % Colours to be used in
Kulite plotting
Avg_size   = 150;                                     % Approximate number of
points wanted in each bin, bins are a constant time size based on this number.
```

```
% Static pressure channels that are used to calibrate the Kulite channels are listed as with the rest in order from front to back

Ps_chan = [46 39 7 8 9 10 43];


% Kulite offsets and plotting of results along a blade chord

Kul_ax_cho = [-80.44 mean([-80.44 -17.62]) -17.62 8.27 34.17 60.06 144.72]+17.62; % Kulite axial positions as a percentage of axial chord starting at the blade leading edge

%Kul_ax_cho = [-63.33 -0.51 25.39 51.28 77.18 161.84]; % Kulite axial positions as a percentage of axial chord starting at the blade leading edge


Kul_offset = [4 5 3 2 1 0 4];                        % Amount of Kulite offset in order from 1 to 6 in terms of number of blades (Kulites at 360/Blade_num apart)

Blade_no   = 22;                                     % Number of rotor blades

%Blade_th   = 19.1;                                   % Angle in theta coordinates from blade leading edge to trailing edge

Blade_th   = 20.;                                   % Angle in theta coordinates from blade leading edge to trailing edge

Blade_th   = pi*Blade_th/180;                        % Converted to radians


% Physical constants

Rho_Hg  = 13550; % Density of mercury [kg/m^3]

g       = 9.81;  % Gravitational constant [m/s^2]

gam_gas = 1.4;   % Gas constant


% Rotor dimensions

Diameter = 11;      % Rotor diameter in inches

Chord    = 0.88824; % Axial Chord in inches


% Contour plot constants

pitch_plot_n = 1;  % Number of pitches to plot

pitch_time_n = 50; % Number of axial lines along the pitch plot

pitch_tang_n = 40; % Number of tangential lines in the axial direction
```

Kulite_rotor_only.m

```
% m-file to plot the Kulite contours for the rotor only case with 7 Kulites

clear all

% Kulite data filename
% 100% Speeds
%Kulite_constants = 'Kulite_constants_PR_1_32_open';   % Full open throttle
without honeycomb PAPER
%Kulite_constants = 'Kulite_constants_PR_1_51';        % Near peak efficiency for
PAPER
%Kulite_constants = 'Kulite_constants_PR_1_66_stall';  % Near stall with honeycomb
PAPER

% 90% Speeds
Kulite_constants = 'Kulite_constants_90_PR_1_49'; % 90% speed near stall
%Kulite_constants = 'Kulite_constants_90_PR_1_38'; % 90% speed near peak
efficiency
%Kulite_constants = 'Kulite_constants_90_PR_1_26'; % 90% speed near choke

% 80% Speeds
%Kulite_constants = 'Kulite_constants_80_PR_1_38'; % 80% speed near stall
%Kulite_constants = 'Kulite_constants_80_PR_1_28'; % 80% speed near peak
efficiency
%Kulite_constants = 'Kulite_constants_80_PR_1_20'; % 80% speed near choke

% 70% Speeds
%Kulite_constants = 'Kulite_constants_70_PR_1_28'; % 70% speed near stall
%Kulite_constants = 'Kulite_constants_70_PR_1_18'; % 70% speed near peak
efficiency
%Kulite_constants = 'Kulite_constants_70_PR_1_15'; % 70% speed near choke

%Kulite_constants
```

```
% Kulite constant file is initialised

%eval(Kulite_constants)


% Raw data is loaded in a seperate function file and also calibrated to make the
analysis function neater

[time,tach,samples,P_PR,m_dot_REF,PR_REF,RPM_sample] =
Load_Kulite_Data(Kulite_constants);


% Function to find the position of the trigger signal, the trigger level and the
Hz frequency of revolution

[Loc,Hz,Trig] = Process_Kulite_Data(tach,samples,time);


% Function to correct the times to phase the Kulites over the blades and correct
for errors in the triggers

[time_phase,time_err,time_angle,P_PR] =
Phase_Kulite_Data(Kulite_constants,Loc,Hz,tach,time,P_PR);


% Function to put all the data into single time traces over ONE ROTATION and also
ONE PASSAGE as if it was sampled at very high speed

[time_rev,P_PR_rev,time_passage,P_PR_passage] =
Rot_Kulite_Data(Kulite_constants,Hz,Loc,time,time_err,time_phase,P_PR);


% Function to find the moving averages of the data to smooth it out (quadratic
function is used to ensure that peak clipping does not occur)

[P_PR_bin,P_PR_bin_DELTA,time_bin] =
Avg_Kulite_Data(Kulite_constants,time_rev,P_PR_rev,Hz,0);


% Function to find the moving averages of the data to smooth it out but for one
averaged blade passage

[P_PR_bin_passage,P_PR_bin_passage_DELTA,time_bin_passage] =
Avg_Kulite_Data(Kulite_constants,time_passage,P_PR_passage,Hz,0);


% Function to interpolate in the axial direction over a single blade passage

[contour_z_passage,contour_th_passage,contour_PR_passage] =
Contour_Kulite_Data(time_bin_passage,P_PR_bin_passage,Hz,time_angle,Kulite_constants);


% Function to interpolate in the axial direction over the entire rotor
```

```
        %[contour_z_rotor,contour_th_rotor,contour_PR_rotor]       =
Contour_Kulite_Data(time_bin,P_PR_bin,Hz,time_angle,Kulite_constants);


        % Data is saved and then loaded so that the whole thing does not have to be run
again

        save Kulite


        %Save_Kulite_Data(Kulite_Subdir,Kulite_Run_no,'save') % Data is saved in raw data
directory

        Save_Kulite_Data('save',Kulite_constants) % Data is saved in raw data directory


        Kulite_figures_rotor_only(Kulite_constants)
```

Load_Kulite_data

```
        % M-function-file to load and calibrate the raw Kulite data

        % This does not use the Kulites around the bottom of the case as they will be used
for the stall cases later


        function [time,tach,samples,P_PR,m_dot_REF,PR_REF,RPM_sample] =
Load_Kulite_Data(Kulite_constants);


        [Kulite_Subdir,Kulite_Run_no,Run_nos,fred1,Kul_no,fred2,fred3,fred4,fred5,fred6,Rh
o_Hg,g,gam_gas,fred7,fred8,fred9,...

                fred10,fred11,Kul_order,Ps_chan] = eval(Kulite_constants);


        % For some reason the Constants file will not spit out more than 20 outputs so
this is inserted here

        Kul_ord_rot_stall = [7 8 9];    % Kulite order of ones installed around the casing
to capture stall cell speed

        Ps_chan_rot_stall = [39 39 39]; % Static pressure channels, the same as all three
Kulites at the same location


        old_dir = pwd;                                          % Current directory is
stored to be returned to later

        new_dir = [old_dir '\' Kulite_Subdir '\' Kulite_Run_no]; % New directory in which
all the data is stored is defined
```

```
        cd(new_dir)                                             % Directory is changed to
one specified

        % File names in the directory are found
        file_info = dir;

        % File names are listed and stored
        Kulite_filenames = [];
        count = 0;
        for j = 1:length(file_info)
            temp = file_info(j).name;
            if file_info(j).isdir == 0 & temp(1) ~= 'K'
                count = count + 1;
                Kulite_filenames = strvcat(Kulite_filenames,file_info(j).name);
                temp =
(max(find(Kulite_filenames(count,:)=='_'))+1):(find(Kulite_filenames(count,:)=='.')-1);
                H_in_Hg(count,:) = str2num(Kulite_filenames(count,temp));
            end % if file_info(j).isdir == 0
            clear temp4
        end % for j = 1:length(file_info)

        % The Kulite files are read in and the mean voltages calculated
        % This is where out of order probes are reordered as the data is streamed in
        for j = 1:size(Kulite_filenames,1)
            if j == 1 % Kulite exposed to atmosphere is used for the data reduction
                Kulite_RawData        = dlmread(Kulite_filenames(j,:),',',5,0); % File is
read in
                Kulite_Rot_Stall      = Kulite_RawData(:,Kul_ord_rot_stall+1);  % Here are
the three Kulites capturing stall
                Kulite_RawData(:,2:8) = Kulite_RawData(:,Kul_order+1);          % Here the
data is sorted into the correct order
                temp                  = Kulite_RawData(:,2:8);                  % Kulite
raw voltage data
                volts_mean            = mean(temp);
                volts_mean_Rot_Stall  = mean(Kulite_Rot_Stall);
```

```
        else % if j == 1
            temp_raw                  = dlmread(Kulite_filenames(j,:),',',5,0); % File
is read in
            volts_mean(j,:)           = mean(temp_raw(:,Kul_order+1));

            % Rotating stall data from the three Kulites are stored to use in the
calibration
            volts_mean_Rot_Stall(j,:) = mean(temp_raw(:,Kul_ord_rot_stall+1));

        end % if j == 1
        clear temp
    end %
    %volts_mean
    %H_in_Hg

    % Static pressures are read in
    cd ..
    file_info = dir; % Get file information of files in the directory

    % Find the largest file
    for j = 1:length(file_info)
        temp(j) = file_info(j).bytes;
    end % for j = 1:length(file_info)
    [Y_max,I_max]    = max(temp);                              % Largest file is
assumed to be the spreadsheet
    Static_RawData   = dlmread(file_info(I_max).name,'\t',1,0);  % File is read in
    m_dot_REF        = mean(Static_RawData(Run_nos,6));          % referred mass flow
rates are pulled out
    PR_REF           = mean(Static_RawData(Run_nos,7));          % referred pressure
ratios are pulled out
    Pressure_RawData = Static_RawData(:,29:76);                  % Pressures are
stripped out

    % Static pressures for the Kulites are inserted using all available runs
```

```
        for i = 1:length(Ps_chan)
            Kulite_P_Static(:,i) = Pressure_RawData(Run_nos,Ps_chan(i));
        end
        Kulite_P_Static  = mean(Kulite_P_Static);

        % Static pressures for the Kulites for the three rotating stall probes
        for i = 1:length(Ps_chan_rot_stall)
            Kulite_P_Stat_rot_stall(:,i) = Pressure_RawData(Run_nos,Ps_chan_rot_stall(i));
        end
        Kulite_P_Stat_rot_stall  = mean(Kulite_P_Stat_rot_stall);

        % Atmospheric pressure is also required in the calibration
        P_atmos = mean([Pressure_RawData(Run_nos,2)]);

        % P infinity at the compressor inlet
        Pt_inf = mean([mean([Pressure_RawData(Run_nos,5)])
mean([Pressure_RawData(Run_nos,6)])]);

        cd(old_dir)   % Directory is changed back to the old one

        % All data is now read in, directory is restored and the calibration is performed
        % Mean pressures are calculated
        for j = 1:size(Kulite_filenames,1)
            dP_mean(j,:)           = Kulite_P_Static - (P_atmos +
Rho_Hg*g*(25.4/1000)*H_in_Hg(j));
            dP_mean_rot_stall(j,:) = Kulite_P_Stat_rot_stall - (P_atmos +
Rho_Hg*g*(25.4/1000)*H_in_Hg(j));
        end % for j = 1:size(Kulite_filenames,1)

        for j = 1:size(volts_mean,2) % Calibration is performed using a linear fit
            [P S MU]     = polyfit(volts_mean(:,j),dP_mean(:,j),1);
            P_dP_v(:,j)  = P';
            S_dP_v(j).S  = S;
            MU_dP_v(:,j) = MU;
```

```
end % for j = 1:size(volts_mean,2)

clear P S MU


for j = 1:size(volts_mean_Rot_Stall,2) % Calibration is performed using a linear
fit for the rotating stall probes

    [P S MU]     = polyfit(volts_mean_Rot_Stall(:,j),dP_mean_rot_stall(:,j),1);

    P_dP_v_rot_stall(:,j)  = P';

    S_dP_v_rot_stall(j).S  = S;

    MU_dP_v_rot_stall(:,j) = MU;

end % for j = 1:size(volts_mean_Rot_Stall,2)

clear P S MU


% Data is sorted into meaningful groups

time       = Kulite_RawData(:,1);   % Kulite time data

volts      = Kulite_RawData(:,2:8); % Kulite raw voltage data

tach       = Kulite_RawData(:,12);  % Kulite trigger voltage

RPM_sample = Kulite_RawData(:,13);  % RPM stated in the data file

samples    = length(time);          % Number of samples


% Pressure signal is processed

for j = 1:size(volts_mean,2)

    P_diff(:,j) = polyval(P_dP_v(:,j)',volts(:,j),S_dP_v(j).S,MU_dP_v(:,j));  %
Differential pressure relative to atmosphere

    P_abs(:,j)  = P_diff(:,j)+P_atmos;                                        %
Absolute pressure

    P_PR(:,j)   = P_abs(:,j)/(gam_gas*Pt_inf);                                %
Pressure as a ratio relative to inlet

end % for j = 1:size(volts_mean,2)


% Polyfit is tested

%for j = 1:size(volts_mean,2)

%    [(polyval(P_dP_v(:,j)',volts_mean(:,j),S_dP_v(j).S,MU_dP_v(:,j)))
dP_mean(:,j)]

%    pause

%end % for j = 1:size(volts_mean,2)
```

```
%dP_mean

%P_dP_v

%S_dP_v

%MU_dP_v


% The pressure calibration is saved so that it can be used in the stall
calculations or elsewhere

save Kulite_calibrate P_dP_v S_dP_v MU_dP_v P_dP_v_rot_stall S_dP_v_rot_stall
MU_dP_v_rot_stall Pt_inf P_atmos
```

test_no_outs.m

```
% m-function file to test the number of outputs possible


function [a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z] = test_no_outs(a)


a = 1

b = 1

c = 1

d = 1

e = 1

f = 1

g = 1

h = 1

i = 1

j = 1

k = 1

l = 1

m = 1

n = 1

o = 1

p = 1

q = 1
```

```
r = 1
s = 1
t = 1
u = 1
v = 1
w = 1
x = 1
y = 1
z = 1
```

Contour_Kulite_data.m

```
% m-funtion to interpolate onto a regular grid from the smoothed data

function [contour_z,old_contour_th,contour_PR] =
Contour_Kulite_Data(time_bin,P_PR_bin,Hz,time_angle,Kulite_constants)

[Kulite_Subdir,Kulite_Run_no,Run_nos,Blade_no,Kul_no,colours,Avg_size,Kul_offset,K
ul_ax_cho,Blade_th,Rho_Hg,g,gam_gas,Diameter,Chord,...
        pitch_plot_n,pitch_time_n,pitch_tang_n] = eval(Kulite_constants);

% Axial grid for the Kulite data is created and scaled to same dimensions as the
pitch
z_bin = ones(size(P_PR_bin,1),1)*Kul_ax_cho;
z_bin = Chord*(z_bin/100)/(pi*Diameter/Blade_no);

% The passage interpolation is performed first
% Grid for the interpolation
contour_z = (min(Kul_ax_cho):(max(Kul_ax_cho)-
min(Kul_ax_cho))/Avg_size:max(Kul_ax_cho));
contour_z = Chord*(contour_z/100)/(pi*Diameter/Blade_no);
contour_z = ones(size(P_PR_bin,1),1)*contour_z;

% The raw data is grouped in larger sections to make sure the edges are correctly
interpolated
```

```
        temp          = max(time_bin(:,1))-min(time_bin(:,1));
        [time_bins I] = unique([time_bin-temp; time_bin; time_bin+temp],'rows');
        z_bins        = [z_bin; z_bin; z_bin];          z_bins    = z_bins(I,:);
        P_PR_bins     = [P_PR_bin; P_PR_bin; P_PR_bin]; P_PR_bins = P_PR_bins(I,:);

        %contour_th = time_bin(:,1)*ones(1,size(P_PR_bin,1));
        contour_th = time_bin(:,1)*ones(1,size(contour_z,2));

        % Interpolation favouring blade
        contour_PR_blade =
griddata(z_bins,time_bins,P_PR_bins,contour_z,contour_th,'cubic');
        disp('1')

        % Interpolation along the passage shock is performed
        % Data is skewed to the actual shape
        time_angle = Blade_no*Hz*time_angle; % Time to physical domain
        time_bin = time_bin + ones(size(P_PR_bin,1),1)*time_angle; % Angle along blades

        % Theta grid is also skewed
        TH_skew    = interp1(z_bin(1,:),time_angle,contour_z)-min(min(time_bin))*0;
        contour_th = contour_th + TH_skew;

        % The raw data is grouped in larger sections to make sure the edges are correctly
interpolated
        temp          = max(time_bin(:,1))-min(time_bin(:,1));
        [time_bins I] = unique([time_bin-temp; time_bin; time_bin+temp],'rows');
        z_bins        = [z_bin; z_bin; z_bin];          z_bins    = z_bins(I,:);
        P_PR_bins     = [P_PR_bin; P_PR_bin; P_PR_bin]; P_PR_bins = P_PR_bins(I,:);

        % Interpolation favouring the passage
        contour_PR_passage =
griddata(z_bins,time_bins,P_PR_bins,contour_z,contour_th,'cubic');
        disp('2')
```

```
% Interpolation along the inlet shock
% Data is skewed along the lines of the inlet shock
temp   = find(Kul_ax_cho<=0);
%time_bin(:,1)+1;
for i = temp
    temp_2               = find(time_bin(:,i)<=(time_bin(1,i)+1));
    [D_PR_max(i) I_max(i)] = min(diff(P_PR_bin(temp_2,i)));   % Doing it with
maximum decrease, better for choke
    %[D_PR_max(i) I_max(i)] = max(diff(P_PR_bin(temp_2,i)));   % Max increase of
gradient better for stall
    %[D_PR_max(i) I_max(i)] = max(P_PR_bin(temp_2,i));
    Z_max(i)             = z_bin(I_max(i),i);
    TH_max(i)            = time_bin(I_max(i),i);
end

TH_max
TH_max(find(TH_max>0)) = TH_max(find(TH_max>0))-1

TH_max                 = [TH_max(1) TH_max(end)]
Z_max                  = [Z_max(1)  Z_max(end)];

%TH_max                 = [TH_max(end-1) TH_max(end)]
%Z_max                  = [Z_max(end-1)  Z_max(end)];

if TH_max(1) < TH_max(2)
    TH_max(1) = TH_max(1)+1
end

Shock_gradient         = polyfit(Z_max,TH_max,1)

old_time_bin           = time_bin;
time_bin               = time_bin - z_bin*Shock_gradient(1);

old_contour_th         = contour_th;
```

```
        contour_th            = contour_th - contour_z*Shock_gradient(1);


        % The raw data is grouped in larger sections to make sure the edges are correctly
interpolated
        temp          = max(time_bin(:,1))-min(time_bin(:,1));
        [time_bins I] = unique([time_bin-temp; time_bin; time_bin+temp],'rows');
        z_bins        = [z_bin; z_bin; z_bin];          z_bins    = z_bins(I,:);
        P_PR_bins     = [P_PR_bin; P_PR_bin; P_PR_bin]; P_PR_bins = P_PR_bins(I,:);


        % Interpolation favouring the upstream shock
        contour_PR_shock =
griddata(z_bins,time_bins,P_PR_bins,contour_z,contour_th,'cubic');
        disp('3')


        % Output contour is composed
        temp = find(contour_z(1,:)<0);
        contour_PR(:,temp) = contour_PR_shock(:,temp);


        % The blade section is separated out
        temp = find(Kul_ax_cho==0);
        [Y I] = max(P_PR_bin(:,temp));


        % Magic number to place the blade relative to the peak pressure on the leading
Kulite
        % This changes from max flow to near stall
        %offset = 0.05; % Offset for peak efficiency
        %offset = 0.2; % Offset for full open throttle
        offset = 0.125; % Offset for full open throttle
        %offset = -0.125; % Offset for full open throttle


        %temp_relax = abs(cos(2*pi*(-offset+old_contour_th(:,temp)-
old_contour_th(I,temp))));
        temp_relax = (cos(pi*(-offset+old_contour_th(:,temp)-old_contour_th(I,temp)))).^4;
        %temp_relax = (sin(pi*(-offset+old_contour_th(:,temp)-
old_contour_th(I,temp)))).^4;
```

```
%[temp_relax old_contour_th(:,temp)-old_contour_th(I,temp)]

%pause

temp = find(contour_z(1,:)>=0);

temp_relax = temp_relax*ones(size(temp));


temp = find(contour_z(1,:)>=0);

contour_PR(:,temp) = (1-temp_relax).*contour_PR_passage(:,temp)+(temp_relax).*contour_PR_blade(:,temp);  % Method using passage and blade


if 0

    max(max(contour_PR_blade))

    max(max(contour_PR_passage))

    max(max(contour_PR_shock))

    max(max(contour_PR))

end


%contour_PR(:,temp) = (1-temp_relax).*contour_PR_shock(:,temp)+(temp_relax).*contour_PR_blade(:,temp);  % Method using the shock and blade

%contour_PR(:,temp) = contour_PR_shock(:,temp);  % Method using only the shock

%contour_PR(:,temp) = contour_PR_blade(:,temp);  % Method using only the blade

%contour_PR(:,temp) = contour_PR_passage(:,temp);  % Method using only the passage
```

fig_contours.m

```
% m function file to produce the moving averages figures plot


function [] = fig_contours(fig_no,contour_z,contour_th,contour_PR,repeat)


%[Kulite_Subdir,Kulite_Run_no,Run_nos,Blade_no,Kul_no,colours,Avg_size,Kul_offset,Kul_ax_cho,Blade_th,Rho_Hg,g,gam_gas,Diameter,Chord,...

%        pitch_plot_n,pitch_time_n,pitch_tang_n] = eval(Kulite_constants);


figure(fig_no);close;figure(fig_no);
```

```
[max(max(contour_PR)) min(min(contour_PR))]


% Contour matrix is constructed to avoid getting lines between each balde row
temp_z = []; temp_th = []; temp_PR = [];
temp = max(contour_th(:,1))-min(contour_th(:,1));
contour_th = contour_th+(-2)*temp;
for i = 1:repeat
    temp_z  = [temp_z;  contour_z];
    temp_th = [temp_th; contour_th+(i-1)*temp];
    temp_PR = [temp_PR; contour_PR];
end % for i = 2:repeat


1.41*[min(min(temp_PR)) max(max(temp_PR))]  % correction for gamma
%contourf(temp_z,temp_th,temp_PR,[.4:.01:1.04])
%contourf(temp_z,temp_th,temp_PR,[.4017:.01:1.0563]) % Peak efficiency
%shading flat
TRI = delaunay(temp_z,temp_th);
trisurf(TRI,temp_z,temp_th,temp_PR*1.41);
shading interp
view(2)


hold on


axis equal
grid on
```

save_mov_avg.m

```
% m-function file to save the raw averaged kulite data


function [] =
save_mov_avg(PR_REF,time_bin_passage,P_PR_bin_passage,Kulite_constants)


% Constants are loaded
```

```
    [Kulite_Subdir,Kulite_Run_no,Run_nos,Blade_no,Kul_no,colours,Avg_size,Kul_offset,K
ul_ax_cho,Blade_th,Rho_Hg,g,gam_gas,Diameter,Chord,...
            pitch_plot_n,pitch_time_n,pitch_tang_n] = eval(Kulite_constants);

    Datafile = [[0 Kul_ax_cho]; [time_bin_passage(:,1) P_PR_bin_passage]];
    temp = num2str(PR_REF);
    filename   = [Kulite_Subdir(1:(min(find(Kulite_Subdir=='%'))-1)) '_1_' temp(3:4)
'.txt'];

    % Directory is changed
    old_dir = pwd;                                      % Current directory is
stored to be returned to later
    if filename(1) == '1'
        new_dir = [old_dir '\' filename(1:3) '%']; % New directory in which all the
data is stored is defined
    else
        new_dir = [old_dir '\' filename(1:2) '%']; % New directory in which all the
data is stored is defined
    end
    cd(new_dir)                                         % Directory is changed to
one specified

    eval(['save ' filename ' Datafile ' '-ascii ' '-double ' '-tabs '])

    % Directory is changed back
    cd(old_dir)   % Directory is changed back to the old one
```

Kulite_figures_rotor_only

```
    % Kulite figures

    function [] = Kulite_figures_rotor_only(Kulite_constants)

    %clear all

    % Kulite data filename
```

```
        %Kulite_constants = 'Kulite_constants_PR_1_32_open';  % Full open throttle without
honeycomb
        %Kulite_constants = 'Kulite_constants_PR_1_49';  % Full open throttle with
honeycomb
        %Kulite_constants = 'Kulite_constants_PR_1_51_open';
        %Kulite_constants = 'Kulite_constants_PR_1_56';  % Full open throttle with
honeycomb
        %Kulite_constants = 'Kulite_constants_PR_1_60';  % Near 85% efficiency
        %Kulite_constants = 'Kulite_constants_PR_1_66_stall';  % Near stall with honeycomb

        %Save_Kulite_Data(Kulite_Subdir,Kulite_Run_no,'load') % Processed data file is
loaded from raw data directory
        Save_Kulite_Data('load',Kulite_constants) % Processed data file is loaded from raw
data directory
        load Kulite

        % Figure of time signal is plotted
        fig_tach_signal(1,time,tach,Loc)

        % Figure of Kulite signals is plotted
        fig_kulite_signal(2,time,P_PR,Loc,Hz,time_err,time_phase,Kulite_constants)

        % Repeating plot of time signals
        figure(3);close;figure(3);
        for i = 1:(length(Loc)-1) % Last incomplete cycle left out
            temp = Loc(i):(Loc(i+1)-1);
            plot((time(temp,1)-time(temp(1),1)-time_err(i)),tach(temp,1),'-b');
            if i == 1
                hold on
            end % if i = 1
        end % for i = 1:(length(Loc)-1)
        clear temp

        % Single trace plots, as if the data was sampled at a very high rate
        fig_mov_avg(4,time_rev,P_PR_rev,[],Hz,Kulite_constants)
```

```
% Moving averages plot over entire row

fig_mov_avg(5,time_bin,P_PR_bin,[],Hz,Kulite_constants)


% Contour plot of passage

fig_contours(6,contour_z_passage,contour_th_passage,contour_PR_passage,6)

temp = axis

axis([-0.5 1 0 2])


% Contour plot of entire rotor

%fig_contours(7,contour_z_rotor,contour_th_rotor,contour_PR_rotor,1)


% Plot of all pressures over 1 blade passage

fig_mov_avg(8,time_passage,P_PR_passage*1.41,[],Hz,Kulite_constants)


% Moving averages plot over single average blade row

fig_mov_avg(9,time_bin_passage,P_PR_bin_passage,[],Hz,Kulite_constants)


% The raw data is saved into text files for comparison with CFD results

save_mov_avg(PR_REF,time_bin_passage,P_PR_bin_passage,Kulite_constants)
```

Avg_Kulite_Data.m

```
% Function to find the moving averages over the entire rotor and then over a
single averages passage


function [P_PR_rev_avg,P_PR_rev_avg_DELTA,time_bin] =
Avg_Kulite_Data(Kulite_constants,time_rev,P_PR_rev,Hz,fred2)


[fred,fred,fred,Blade_no,Kul_no,fred,Avg_size,fred,fred,fred,fred,fred,fred,fred,f
red,fred,fred,fred,fred,fred] = eval(Kulite_constants);


Avg_time_factor = 1/(max(max(time_rev))*Hz);


if max(max(time_rev)) > Blade_no  % An entire passage

    bins = 0:1/Avg_size:Blade_no;
```

```
        else % if max(max(time_rev)) > Blade_no

            bins = 0:1/Avg_size:1;

        end  % if max(max(time_rev)) > Blade_no


        % Memory is assigned

        time_bin           = zeros(length(bins),length(Kul_no));

        P_PR_rev_avg       = time_bin;

        P_PR_rev_avg_DELTA = time_bin;


        % Moving average over all the passages

        for j = Kul_no

            home

            j

            % Data bin ends, this method is an attempt to be faster than using the brute
force 'find' function (save huge amounts of time)

            Data_i_low  = 1; % Initial bin beggining

            Data_i_high = 2; % Initial bin end


            tic

            for i = 1:length(bins)


                % Start and end times of each bin

                %bin_start = bins(i)-(1/Avg_size)/2;   bin_end   = bins(i)+(1/Avg_size)/2;

                bin_start = bins(i)-6*(1/Avg_size);   bin_end   = bins(i)+6*(1/Avg_size);
% Use a bigger bin


                Data_i_low_old = Data_i_low; Data_i_high_old = Data_i_high; % Old bin
borders to check if polyfit needs to be done (eliminates repeating of calculations)


                % Index of lower side of the bin

                while time_rev(Data_i_low,j) < bin_start

                    [time_rev(Data_i_low,j) bin_start];

                    Data_i_low = Data_i_low + 1;

                end
```

```
            % Index of high side of the bin
            while time_rev(Data_i_high,j) < bin_end
                [time_rev(Data_i_high,j) bin_end];
                Data_i_high = Data_i_high + 1;
            end

            Data_i = (Data_i_low:(Data_i_high-1))';

            % Check to make sure that a sufficient number of points is available for
the interpolation
            if length(Data_i) < 4
                Data_i = ((Data_i_high-4):(Data_i_high-1))';
            end

            % A quadratic moving average is fitted through the data, this smoothes the
data without clipping the peaks (0 = mean, 1 = linear)
            if length(unique_cmtfm(time_rev(Data_i,j),'rows',1e-10)) > 2
                [P,S,MU] = polyfit(time_rev(Data_i,j),P_PR_rev(Data_i,j),2); % Fit is
only performed if the bin has changed

                [P_PR_rev_avg(i,j) P_PR_rev_avg_DELTA(i,j)] = polyval(P,bins(i),S,MU);
% Quadratic moving average and 50% certainty interval
                time_bin(i,j)                               = bins(i);
% Time of bin
            elseif length(unique_cmtfm(time_rev(Data_i,j),'rows',1e-10)) == 2
                [P,S,MU] = polyfit(time_rev(Data_i,j),P_PR_rev(Data_i,j),1); % Fit is
only performed if the bin has changed

                [P_PR_rev_avg(i,j) P_PR_rev_avg_DELTA(i,j)] = polyval(P,bins(i),S,MU);
% Quadratic moving average and 50% certainty interval
                time_bin(i,j)                               = bins(i);
% Time of bin

            else % length(unique_cmtfm(time_rev(Data_i,j),'rows',1e-10)) > 2
                [P,S,MU] = polyfit(time_rev(Data_i,j),P_PR_rev(Data_i,j),0); % Fit is
only performed if the bin has changed
```

```
                    [P_PR_rev_avg(i,j) P_PR_rev_avg_DELTA(i,j)] = polyval(P,bins(i),S,MU);
% Quadratic moving average and 50% certainty interval
                    time_bin(i,j)                               = bins(i);
% Time of bin
                end  % if (max(time_rev(Data_i,j))-min(time_rev(Data_i,j)))>1e-10
                if fred2
                    [i j]
                    [time_rev(Data_i,j) P_PR_rev(Data_i,j)]
                    unique_cmtfm(time_rev(Data_i,j),'rows',1e-10)
                    P_PR_rev_avg(i,j)
                    pause
                end



            end % for i = 1:size(time_rev,1)
        end % for j = Kul_no
```

Rot_Kulite_Data.m

```
        % m function file to place the long sample into a single short very high speed
sample


        function [time_rev,P_PR_rev,time_passage,P_PR_passage] =
Rot_Kulite_Data(Kulite_constants,Hz,Loc,time,time_err,time_phase,P_PR)


        [fred,fred,fred,Blade_no,Kul_no,fred,fred,fred,fred,fred,fred,fred,fred,fred,fred,
fred,fred,fred,fred,fred] = eval(Kulite_constants);


        % Time and pressures from hair plots are stored and sorted too
        time_rev = []; P_PR_rev = [];
        for j = Kul_no
            temp_time_rev = [];
            temp_P_PR_rev = [];
            for i = 1:(length(Loc)-1) % Last incomplete cycle left out
                temp      = Loc(i):(Loc(i+1)-1);                                      %
Indicies of elements in the relevant cycle
```

```
            % A small amount of data is added to the beggining and end of the sample
to make the moving averages correct (1 blade pitch)

            temp_nose = temp(1)-round(length(temp)/Blade_no);

            temp_tail = temp(end) + round(length(temp)/Blade_no);

            temp2     = temp_nose:temp_tail; % Total data


            temp_time = (time(temp2,1)-time(temp(1),1)-time_err(i)-time_phase(j)); %
Actual time elements

            temp_P_PR = P_PR(temp2,j);                                            %
Actual pressure elements

            % Time and pressure ratios on a single time axis are stored

            temp_time_rev = [temp_time_rev; temp_time];

            temp_P_PR_rev = [temp_P_PR_rev; temp_P_PR];

        end % for i = 1:(length(Loc)-1)


        % Continuous signal

        time_rev(:,j) = temp_time_rev; clear temp_time_rev;

        P_PR_rev(:,j) = temp_P_PR_rev; clear temp_P_PR_rev;


        % Signals are now sorted as if they all stream in one after another

        [time_rev(:,j) I_time_rev] = sort(time_rev(:,j));    % Sorted time signal and
indicies

        P_PR_rev(:,j)              = P_PR_rev(I_time_rev,j); % Sorted PR signal using
the time indicies

    end % for j = Kul_no


    % Data grouped over one blade passage

    Passage_times = 0:(1/Hz)/Blade_no:(1/Hz);

    time_passage = []; P_PR_passage = [];


    for j = Kul_no

        temp_time_passage = [];

        temp_P_PR_passage = [];
```

```
            % Each blade passage is done
            for k = 1:Blade_no
                half_passage_time = (Passage_times(k+1)-Passage_times(k))/10;  % 1/10 a
passage length to be added to each side of the sample
                temp              = find(time_rev(:,j)>=(Passage_times(k) -
half_passage_time) & time_rev(:,j)<(Passage_times(k+1)+half_passage_time)); % Times in
passage
                temp_time_passage = [temp_time_passage; time_rev(temp,j)-
Passage_times(k)]; % Times are added to the passage
                temp_P_PR_passage = [temp_P_PR_passage; P_PR_rev(temp,j)];
            end % for k = 1:Blade_no

            % Data is sorted
            [temp_time_passage I_temp_time_passage] = sort(temp_time_passage);
            temp_P_PR_passage                       =
temp_P_PR_passage(I_temp_time_passage);
            [j length(temp_P_PR_passage)]

            % Data is grouped in seperate Kulite columns but all kept to the same length
but clipping is actually very very small
            if length(time_passage) ~= 0
                min_length   = min([length(time_passage) length(temp_time_passage)]);

                time_passage      = time_passage(1:min_length,:);
                time_passage(:,j) = temp_time_passage(1:min_length);

                P_PR_passage      = P_PR_passage(1:min_length,:);
                P_PR_passage(:,j) = temp_P_PR_passage(1:min_length);

            else % first time around
                time_passage(:,j) = temp_time_passage;
                P_PR_passage(:,j) = temp_P_PR_passage;
            end
        end % for j = Kul_no
```

```
% Time data is non-dimensionalised so that each blade passage is unity long
time_rev     = Blade_no*Hz*time_rev; % Total revolution
time_passage = Blade_no*Hz*time_passage;      % One average passage
```

fig_mov_avg.m

```
% m function file to produce the moving averages figures plot


function [] =
fig_mov_avg(fig_no,time_rev,P_PR_rev_avg,P_PR_rev_avg_DELTA,Hz,Kulite_constants)


[Kulite_Subdir,Kulite_Run_no,Run_nos,Blade_no,Kul_no,colours,Avg_size,Kul_offset,K
ul_ax_cho,Blade_th,Rho_Hg,g,gam_gas,Diameter,Chord,...
        pitch_plot_n,pitch_time_n,pitch_tang_n] = eval(Kulite_constants);


figure(fig_no);close;figure(fig_no);


for j = Kul_no
%for j = 5
    temp      = find(P_PR_rev_avg(:,j)~=0);
    temp_time = time_rev(temp,j)+0*Hz*Blade_no;              % Non dimensionalised
according to the pitch
    plot(temp_time,P_PR_rev_avg(temp,j),['-' colours(j)]);
    if j == Kul_no(1)
        hold on; grid on
    end % if j == Kul_no(1)
    if isempty(P_PR_rev_avg_DELTA)
    else % if isempty(P_PR_rev_avg_DELTA)
        plot(temp_time,P_PR_rev_avg(temp,j)+P_PR_rev_avg_DELTA(temp,j),[':'
colours(j)]);
        plot(temp_time,P_PR_rev_avg(temp,j)-P_PR_rev_avg_DELTA(temp,j),[':'
colours(j)]);
    end % if isempty(P_PR_rev_avg_DELTA)
end % for j = Kul_no


temp = axis;
if temp(2) < 22
```

```
        axis([0 1 temp(3) temp(4)])
    end % if temp(2) < 22
```

fig_cont_raw.m

```
    % m function file to produce the moving averages figures plot from the raw data


    function [] =
fig_contours(fig_no,time_passage,time_angle,P_PR_passage_avg,Hz,Kulite_constants)


    [Kulite_Subdir,Kulite_Run_no,Run_nos,Blade_no,Kul_no,colours,Avg_size,Kul_offset,K
ul_ax_cho,Blade_th,Rho_Hg,g,gam_gas,Diameter,Chord,...
            pitch_plot_n,pitch_time_n,pitch_tang_n] = eval(Kulite_constants);


    figure(fig_no);close;figure(fig_no);


    % At this point the data is set out as if the Kulites are set in a line, only some
of the points are needed as the data is so fine


    % Data needs to be trimmed, trimming starts and ends at the same points
    for j = Kul_no
        temp_start(j) = max(find(P_PR_passage_avg(1:round(end/2),j)==0));
        temp_end(j)   =
min(find(P_PR_passage_avg(round(end/2):end,j)==0))+round(length(P_PR_passage_avg(:,j))/2)
-2;
        %interp1(time_passage(:,j),P_PR_passage_avg(:,j),[time_start time_end])
    end % for j = Kul_no
    temp_start = max(temp_start(Kul_no)); temp_end   = min(temp_end(Kul_no));


    P_PR_passage_avg = P_PR_passage_avg(temp_start:temp_end,:);
    time_passage     = time_passage(temp_start:temp_end,:);


    % Conversion of data from the time to the physical domain
    time_passage_pitch   =
Blade_no*(time_passage+ones(size(time_passage,1),1)*time_angle)*Hz; % Also offset of
chord passages is added
    Kul_ax_cho_pitch     = Chord*(Kul_ax_cho/100)/(pi*Diameter/Blade_no);
% Scaling
```

```
        Kul_ax_cho_pitch     = ones(length(time_passage_pitch),1)*Kul_ax_cho_pitch;
% Rectangular grid for plotting contours


        temp_x = time_passage_pitch;

        temp_y = Kul_ax_cho_pitch;

        temp_z = P_PR_passage_avg;


        Red_fact = Blade_no; % factor by which the number of points in reduced, based on
the number of blades for simplicity


        % Points are reduced by some skilled and cunning coding

        temp = temp_x(1:Red_fact:end,:); temp_x = [temp; temp_x(end,:)];

        temp = temp_y(1:Red_fact:end,:); temp_y = [temp; temp_y(end,:)];

        temp = temp_z(1:Red_fact:end,:); temp_z = [temp; temp_z(end,:)];


        temp = 1;

        %for i = -2:3

        for i = 1

            contour(temp_y(:,Kul_no),temp_x(:,Kul_no)+i,temp_z(:,Kul_no))

            if temp

                hold on; temp = 0;

            end

        end

        axis equal


        % Blade leading and trailing edges are drawn in

        axis([-1 2 0 3])

        temp   = axis;

        temp_x = [temp(3:4); temp(3:4)]';

        temp_y = [0 0; Chord*(100/100)/(pi*Diameter/Blade_no)
Chord*(100/100)/(pi*Diameter/Blade_no)]';

        plot(temp_y,temp_x)
```

fig_kulite_signal.m

```
        % m function file to plot the tach signal and Loc points


        function [] =
fig_kulite_signal(fig_no,time,P_PR,Loc,Hz,time_err,time_phase,Kulite_constants)


        [Kulite_Subdir,Kulite_Run_no,Run_nos,Blade_no,Kul_no,colours,Avg_size,Kul_offset,K
ul_ax_cho,Blade_th,Rho_Hg,g,gam_gas,Diameter,Chord,...
                pitch_plot_n,pitch_time_n,pitch_tang_n] = eval(Kulite_constants);


        % Repeating plot of kulite signals

        % Also time and pressures from hair plots are stored and sorted too

        figure(fig_no);close;figure(fig_no);

        for j = Kul_no

            for i = 1:(length(Loc)-1) % Last incomplete cycle left out

                temp      = Loc(i):(Loc(i+1)-1);                                      %
Indicies of elements in the relevant cycle

                temp_time = (time(temp,1)-time(temp(1),1)-time_err(i)+time_phase(j)); %
Actual time elements

                temp_P_PR = P_PR(temp,j);                                             %
Actual pressure elements

                plot(temp_time,temp_P_PR,['-' colours(j)]);

                if j == Kul_no(1)

                    %        axis([0 1.4e-4 -.15 .3])

                    hold on

                    grid on

                end % if i = 1

            end % for i = 1:(length(Loc)-1)


            % Mean pressure from Kulite is plotted

            temp = axis;

            plot([temp(1) temp(2)],[mean(P_PR(:,j)) mean(P_PR(:,j))],['-' colours(j)]);


            % Offset is plotted
```

```
            plot([Kul_offset(j)/(Blade_no*Hz) Kul_offset(j)/(Blade_no*Hz)],[0 1],['-' colours(j)])
        end % for j = Kul_no
        clear temp

        % Figure is modified
        title(['Kulite ' num2str(Kul_no)])
        axis auto
        temp = axis;
        axis([0 temp(2) 0.4 1.1])
        clear temp
```

Phase_Kulite_Data.m

```
        % m-function to calculate the amount the Kulite probes need to be phased in order to lie along a blade.

        function [time_phase,time_err,time_angle,P_PR] = Phase_Kulite_Data(Kulite_constants,Loc,Hz,tach,time,P_PR)

        % Kulite constants are loaded
        [fred,fred,fred,Blade_no,Kul_no,fred,fred,Kul_offset,Kul_ax_cho,Blade_th,fred,fred,fred,fred,fred,fred,fred,fred,fred,fred] = eval(Kulite_constants);

        Trig = mean([min(tach) max(tach)]);

        % Kulites are now lined up as if they sample along the blade chord, this will reduce the amount of signal clipping needed
        %[(Kul_ax_cho*Blade_th/100)/(2*pi*Hz); Kul_offset/(Blade_no*Hz); +Kul_offset/(Blade_no*Hz)+(Kul_ax_cho*Blade_th/100)/(2*pi*Hz)]

        time_angle = (Kul_ax_cho*Blade_th/100)/(2*pi*Hz); % Time that needs to be trimmed from the samples to line them over blade
        time_trim  = Kul_offset/(Blade_no*Hz);            % Time to line the Kulites up along a straight line
        time_trim  = time_trim + time_angle;              % Two times are combined to make it simpler to work with
        time_trim  = time_trim - min(time_trim);
```

```
% Kulite signals are trimmed to get all the signals in phase. RPM is assumed
constant over sample period

for j = Kul_no

    temp          = find(time>=time_trim(j)); % Number of points that are before
the correct sampling time

    time_phase(j) = [time(temp(1))-time_trim(j)]; % There is a slight error
associated with the chop off

    P_PR(:,j)    = [P_PR(temp,j); zeros(length(P_PR(:,j))-(length(temp)),1)];

end % for j = Kul_no


% RPM from each trigger pulse is calculated and error according to the deviation
from the mean RPM is calculated

time_err       = -[time(Loc)-[time(Loc(1)):(time(Loc(end))-
time(Loc(1)))/(length(Loc)-1):time(Loc(end))]'];
```

Save_Kulite_Data.m

```
% M-file to save the processed data into the raw data file so that the whole
process does not need to be repeated


function [] = Save_Kulite_Data(load_or_save,Kulite_constants)


[Kulite_Subdir,Kulite_Run_no,Run_nos,Blade_no,Kul_no,colours,Avg_size,Kul_offset,K
ul_ax_cho,Blade_th,Rho_Hg,g,gam_gas,Diameter,Chord,...
        pitch_plot_n,pitch_time_n,pitch_tang_n] = eval(Kulite_constants);


if load_or_save == 'save'

    load Kulite                                                  % Data from current
directory is loaded into the present function

    old_dir = pwd;                                               % Current directory
is stored to be returned to later

    new_dir = [old_dir '\' Kulite_Subdir '\' Kulite_Run_no]; % New directory in
which all the data is stored is defined

    cd(new_dir)                                                  % Directory is
changed to one specified


    % Data is saved in the raw data directory

    save Kulite
```

```
        cd(old_dir)   % Directory is changed back to the old one

    end % if load_or_save == 'save'


    if load_or_save == 'load'

        old_dir = pwd;                                              % Current directory
is stored to be returned to later

        new_dir = [old_dir '\' Kulite_Subdir '\' Kulite_Run_no]; % New directory in
which all the data is stored is defined

        cd(new_dir)                                                 % Directory is
changed to one specified


        load Kulite                                                 % Data from raw data
directory is loaded into the present function


        cd(old_dir)                                                 % Directory is
changed back to the old one

        save Kulite                                                 % Data is saved into
the current directory


    end % if load_or_save == 'load'
```

Process_Kulite_Data.m

```
    % m-function file to process the Kulite raw data


    function [Loc,Hz,Trig] = Process_Kulite_Data(tach,samples,time);


    Trig = mean([min(tach) max(tach)]);


    % Location of trigger points and correction to exact trigger timing point

    Loc = find( tach(2:samples)<Trig & tach(1:samples-1)>Trig ); %location of time of
start of rev


    % If Loc (location) if at the beggining of the sample it is discarded

    if Loc(1) < 3

        Loc = Loc(2:end);

    end
```

```
% Last trigger is discarded to ensure trailing zeros resulting from probe lining
up do not effect the calculations.
Loc = Loc(1:(end-1));


% Frequency of rotor revolution over the sample period
Hz = (length(Loc)-1)/(time(Loc(end))-time(Loc(1)));
```

unique_cmtfm.m

```
function [b,ndx,pos] = unique_cmtfm(a,flag,tol)
% This also finds points that are within a certain tolerance of each other
%UNIQUE Set unique.
%   UNIQUE(A) for the array A returns the same values as in A but
%   with no repetitions.  A will also be sorted.  A can be a cell
%   array of strings.
%
%   UNIQUE(A,'rows') for the matrix A returns the unique rows of A.
%
%   [B,I,J] = UNIQUE(...) also returns index vectors I and J such
%   that B = A(I) and A = B(J) (or B = A(I,:) and A = B(J,:)).
%
%   See also UNION, INTERSECT, SETDIFF, SETXOR, ISMEMBER.

%   Copyright 1984-2000 The MathWorks, Inc.
%   $Revision: 1.21 $  $Date: 2000/06/01 04:40:02 $

%   Cell array implementation in @cell/unique.m

if nargin == 2
    tol = 0;
end

if nargin==1 | isempty(flag),
  % Convert matrices and rectangular empties into columns
  if length(a) ~= prod(size(a)) | (isempty(a) & any(size(a)))
```

```
    a = a(:);
  end
  b = a;
  ndx = (1:length(a))';
  %  [b,ndx] = sort(a);
  % d indicates the location of matching entries
  %%d = b((1:end-1)')==b((2:end)');
  d = abs(b((1:end-1)')-b((2:end)') < tol);
  b(find(d)) = [];
  if nargout==3, % Create position mapping vector
    pos = zeros(size(a));
    pos(ndx) = cumsum([1;~d(:)]);
  end
else
  if ~isstr(flag) | ~strcmp(flag,'rows'), error('Unknown flag.'); end
  b = a;
  ndx = (1:size(a,1))';
  %  [b,ndx] = sortrows(a);
  [m,n] = size(a);
  if m > 1 & n ~= 0
    % d indicates the location of matching entries
    %%d = b(1:end-1,:)==b(2:end,:);
    d = abs(b(1:end-1,:)-b(2:end,:)) < tol;
  else
    d = zeros(m-1,n);
  end
  d = all(d,2);
  b(find(d),:) = [];
  if nargout==3, % Create position mapping vector
    pos(ndx) = cumsum([1;~d]);
    pos = pos';
  end
end
```

```
ndx(find(d)) = [];
```

fig_tach_signal.m

```
% m function file to plot the tach signal and Loc points


function [] = fig_tach_signal(fig_no,time,tach,Loc)


figure(fig_no);close;figure(fig_no);

plot(time(:,1),tach(:,1)); hold on;

plot(time(Loc,1),tach(Loc,1),'+r')
```

APPENDIX C: MATLAB M-FILES (STALL CASES)

calibrate_data.m

```
% m-file to calibrate the pressure data from volts to a pressure ratio

% The last measured inlet pressure is used which may not be correct.

% Look at the upstream static pressure as this may lead to some clues about the mass flow rate.


function [PR,PR_rot_stall] = calibrate_data(volts)


% Constants are read in

[Kul_order,Kul_offset,Blade_no,Blade_th,gam_gas,fred,fred,fred,fred,fred,Kul_ord_rot_stall] = ...

    Kulite_constants_stall_90;


% Kulite calibration data from the run closest to stall is read in

% This is in the order of the Kulites from the front to the rear

load Kulite_calibrate_90_stall


% Pressure signal is processed

for j = 1:length(Kul_order)

    P_diff(:,j)        = polyval(P_dP_v(:,j)',volts(:,Kul_order(j)),S_dP_v(j),MU_dP_v(:,j)); % Differential pressure relative to atmosphere

    P_abs(:,j)         = P_diff(:,j)+P_atmos;                                             % Absolute pressure

    %PR(:,Kul_order(j)) = P_abs(:,j)/(gam_gas*Pt_inf);                                   % Pressure as a ratio relative to inlet

    PR(:,Kul_order(j)) = P_abs(:,j)/(1*Pt_inf);                                          % Pressure as a ratio relative to inlet

end % for j = 1:length(Kul_order)

clear P_diff P_abs
```

```
% Pressure signal for the three transducers that captured the rotating stall speed
for j = 1:length(Kul_ord_rot_stall)
    P_diff(:,j)        =
polyval(P_dP_v_rot_stall(:,j)',volts(:,Kul_ord_rot_stall(j)),...
        S_dP_v_rot_stall(j),MU_dP_v_rot_stall(:,j));   % Differential pressure
relative to atmosphere
    P_abs(:,j)         = P_diff(:,j)+P_atmos;          % Absolute pressure
    %PR(:,Kul_order(j)) = P_abs(:,j)/(gam_gas*Pt_inf); % Pressure as a ratio
relative to inlet
    PR_rot_stall(:,Kul_ord_rot_stall(j)) = P_abs(:,j)/(1*Pt_inf);         %
Pressure as a ratio relative to inlet
end % for j = 1:length(Kul_ord_rot_stall)
```

countour_data.m

```
% m-function file to interpolate the Kulite data onto a finer grid and perform the
required transforms
% to get a smooth picture of the pressure through the rotor.

function [PR_X,PR_Y,PR_Z] =
contour_data(PR,time,N_RPM,N_pitch,N_grid,pitch_offset)

% Constants are read in
[Kul_order,Kul_offset,Blade_no,Blade_th,gam_gas,Kul_ax_cho,Diameter,Chord,shock,SC
F] = Kulite_constants_stall_90;

Blade_no_SCF = ceil(Blade_no/SCF);  % The number of blade pitches needed to
complete the interpolation.

% The axial Kulite positions are scaled to the size of the pitch
Cho_pit    = Chord/(pi*Diameter/Blade_no); % Axial chord / Physical size of blade
pitch
Kul_ax_pit = Cho_pit*(Kul_ax_cho/100);     % Non-dimensional Kulite axial chord
relative to pitch length

% Underlying Kulite grid is set up
```

```
    for j = 1:length(Kul_order)
        K_Z(:,j) = PR(:,Kul_order(j));                % Pressure ratio grid
        K_X(:,j) = Kul_ax_pit(j)*ones(size(K_Z(:,j))); % Axial chord grid
        K_Y(:,j) = time(:,Kul_order(j));              % Tangential grid
    end % for j = 1:length(Kul_order)

    % Refined grid over the required rotation is set up
    PR_X = [Kul_ax_pit(1) Kul_ax_pit(end)];                          % X or axial
coordinates
    PR_X = PR_X(1):(PR_X(2)-PR_X(1))/(N_grid-1):PR_X(2);
    PR_X = ones(Blade_no_SCF*N_grid,1)*PR_X;

    % Y or tangential coordinates at desired rotation
    PR_Y = N_pitch+[(N_RPM-1)*Blade_no ((N_RPM-1)*Blade_no+Blade_no_SCF)];
    PR_Y = (PR_Y(1):(PR_Y(2)-PR_Y(1))/(Blade_no_SCF*N_grid-1):PR_Y(2))';
    PR_Y = PR_Y*ones(1,N_grid);

    % This is the first interpolation with the blades in their normal positon
    PR_Z_passage = griddata(K_X,K_Y,K_Z,PR_X,PR_Y,'cubic');

    % This is the second interpolation with the grid skewed axially along the blades
    % Both the interpolation and underlying grid are skewed and then the interpolation
grid is skewed back

    % Arc of blade as fraction of pitch and non-dimensionalised wrt to pitch
    temp = ((Blade_th)/(2*pi/Blade_no))/Cho_pit;

    % Grids are skewed
    PR_Y_blade = PR_Y - temp*PR_X;
    K_Y_blade  = K_Y  - temp*K_X;

    % Interpolation along the blade is performed
    PR_Z_blade = griddata(K_X,K_Y_blade,K_Z,PR_X,PR_Y_blade,'cubic');
```

```
        % This is the 3rd part of the interpolation along the bow shock upstream of the
rotor
        PR_Y_shock = PR_Y + PR_X*tan(shock);
        K_Y_shock  = K_Y  + K_X*tan(shock);

        % Interpolation along the shock is performed
        PR_Z_shock = griddata(K_X,K_Y_shock,K_Z,PR_X,PR_Y_shock,'cubic');

        % Different parts of the three interpolations are now pieced together
        PR_Z = PR_Z_shock;

        % Upstream of rotor
        temp         = find(PR_X(1,:) >= 0);

        % A cos function is used for the combining of the two sets
        relax = PR_Y_blade-PR_Y_blade(1,1)-pitch_offset; relax = cos(pi*(relax)).^4;

        % Within the rotor row
        temp         = find(PR_X(1,:) >= 0);
        PR_Z(:,temp) = relax(:,temp).*PR_Z_blade(:,temp) + (1-
relax(:,temp)).*PR_Z_passage(:,temp);

        % Downstream of the rotor the normal passage interpolation is used but this is
probably not
        % physically realistic as a wake does exist
        %temp         = find(PR_X(1,:) >= Cho_pit);
        %PR_Z(:,temp) = PR_Z_blade(:,temp);

        %figure(6); close; figure(6)
        %contourf(PR_X,PR_Y,relax)

        count = 1;
        for i = PR_Y(1,1):(PR_Y(1,1)+Blade_no)
```

```
            temp_1 = find(floor(PR_Y(:,1))==i);              % Data points that fall in
the local pitch set

            % This is where the stall cell is at the end of the pitch
            temp_2  = i+(1-SCF)*count;                          % Pitch that the stall
cell data should come from

            temp_3  = find(floor(PR_Y(:,1)) == floor(temp_2));  % Data points from leading
pitch set
            temp_4  = temp_3+length(temp_3);                    % Data points from
trailing pitch set

            fract_1 = 1-(temp_2-floor(temp_2));                 % Fraction from the
leading pitch set

            % This is where the stall cell is at the beggining of the pitch
            temp_5  = i+(1-SCF)*(count-1);                      % Pitch that the stall
cell data should come from

            temp_6  = find(floor(PR_Y(:,1)) == floor(temp_5));  % Data points from leading
pitch set
            temp_7  = temp_6+length(temp_6);                    % Data points from
trailing pitch set

            fract_2 = 1-(temp_5-floor(temp_5));                 % Fraction from the
leading pitch set

            % Linear distribution between leading and trailing edge
            spread = (0:1/max([1 (length(temp_1)-1)]):1)'*ones(1,length(PR_Y(1,:)));

            if size(temp_1) == size(temp_3)
                PR_Z(temp_1,:) = spread.*(fract_1*PR_Z(temp_3,:) + (1-
fract_1)*PR_Z(temp_4,:)) +...% Trailing edge of pitch
                    (1-spread).*(fract_2*PR_Z(temp_6,:) + (1-fract_2)*PR_Z(temp_7,:)); %
Leading edge of pitch cell
            end
```

```
            count = count+1; % Internal counter to tell which cell is being modified in
the local set

        end % for i = PR_Y(1,1)

        % Grid is trimmed to one revolution
        temp  = (find(PR_Y(:,1) <= ceil(Blade_no + min(PR_Y(:,1)) )) );
        temp  = [temp; max(temp)+1 ];
        PR_X = PR_X(temp,:);
        PR_Y = PR_Y(temp,:);
        PR_Z = PR_Z(temp,:);
```

Kulite_constants_stall_90.m

```
        % m-function file to store the Kulite constants

        function
[Kul_order,Kul_offset,Blade_no,Blade_th,gam_gas,Kul_ax_cho,Diameter,Chord,shock,SCF,Kul_o
rd_rot_stall] = ...
            Kulite_constants_stall_100()

        % In one of the tests the Kulites were mixed up so this step was introduced in
case it happens again
        Kul_order  = [1 9 2 3 4 5 6];

        % These are the Kulites arranged to be able to capture the speed of the rotating
stall
        Kul_ord_rot_stall = [7 8 9];

        % Amount of Kulite offset in order from 1 to 6 in terms of number of blades
(Kulites at 360/Blade_num apart)
        Kul_offset = [4 5 3 2 1 0 4];

        % Kulite offsets and plotting of results along a blade chord
```

```
        Kul_ax_cho = [-80.44 mean([-80.44 -17.62]) -17.62 8.27 34.17 60.06 144.72]+17.62;
% Kulite axial positions as a percentage of axial chord starting at the blade leading
edge

        Blade_no   = 22;                 % Number of rotor blades
        %Blade_th   = 19.1;              % Angle in theta coordinates from blade leading
edge to trailing edge
        Blade_th   = 20.;                % Angle in theta coordinates from blade leading
edge to trailing edge
        Blade_th   = pi*Blade_th/180;    % Converted to radians

        % Physical constants
        gam_gas = 1.4;   % Gas constant

        % Rotor dimensions
        Diameter = 11;      % Rotor diameter in inches
        Chord    = 0.88824; % Axial Chord in inches

        % The angle of the shock in radians relative to axial
        shock = 10;
        shock = shock*(pi/180);

        % Stall cell frequency as a fraction of the RPM
        %SCF = 0.82;  % Pre-stall
        %SCF = 0.82;  % 1 Rev
        %SCF = 0.82;  % 2 Rev
        %SCF = 0.75;  % 3 Rev
        %SCF = 0.70;  % 4 Rev
        %SCF = 0.70;  % 5 Rev
        %SCF = 0.66;  % 6 Rev
        SCF = 0.66;  % 50 Rev
        %Stall90
```

Load_data_90.m

```
% 90% speed
% M-file to pull in stall data and plot it out

clear all
close all
pack

% Constants
RPM_design = 27085;         % Design RPM in RPM
RPM_Hz     = RPM_design/60; % Design RPM in Hz
N_mov_avg  = 5;             % Number of times the moving average is performed

% Stall data is loaded
%Raw_data = dlmread('Dx2004_0914_1001_90_stall.csv',',','A187500..M200000'); %
Stall cell inception figure
Raw_data = dlmread('Dx2006_0403_1143.csv',',','A203000..M219000'); % Contour plot
data

% The data is seperated and initial processing is done
[time,volts,tach,Loc,RPM,time_err] = Process_data(Raw_data,N_mov_avg);

% Data needs to be normalised, each blade pitch is equal to unity
[time] = time_to_pitch(time,Loc,time_err);

% Data needs to be phased to correct for Kulite offset, this function puts them
all in a straight line
% The time vector becomes a matrix
[time,volts] = phase_data(time,volts);
%[time] = phase_data(time,volts);

% Data needs to be calibrated to pressure
[PR,PR_rot_stall] = calibrate_data(volts);
```

```
% Data is interpolated from the Kulites onto a finer grid
%N_RPM   = 3;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 5;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 6;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 7;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 9;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 10;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 12;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 13;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 15;  % Rotation number that the data will be interpolated from, INTEGER
%N_RPM   = 16;  % Rotation number that the data will be interpolated from, INTEGER
N_RPM   = 50;  % Rotation number that the data will be interpolated from, INTEGER


%N_pitch = 0;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = -5;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = 0;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = 7;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = -7;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = 2;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = -9;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = 2;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = -10;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
%N_pitch = 0;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
N_pitch = 0;   % Blade pitch number, basically the fraction of rotation that is
needed, INTEGER
```

```
        pitch_offset = 0.1; % Amount of offset to use to ensure that the interpolation
function lies along the blade, 0-1 REAL

        N_grid  = 100; % Number of grid points in each blade pitch, INTEGER


        % The RPM is plotted

        Plot_rpm(4,RPM,RPM_Hz)


        [PR_X,PR_Y,PR_Z] = contour_data(PR,time,N_RPM,N_pitch,N_grid,pitch_offset);


        % A particular column of the data is plotted

        % Upstream probes

        offset = -60; % Offset to start count at zero

        offset = [offset offset-7-(1/3) offset-12-(2/3)]; % offsets to get same blades
passing at the same time

        Plot_data(1,1,'k',offset(1)+time,PR_rot_stall,Loc,time_err,0,7,1,3)

        Plot_data(1,0,'b',offset(2)+time,PR_rot_stall,Loc,time_err,0,8,2,3)

        Plot_data(1,0,'r',offset(3)+time(:,7)*ones(1,size(time,2)),PR_rot_stall,Loc,time_e
rr,0,9,3,3) % The time offset is not needed


        % Blade passage probes

        Plot_data(2,1,'b',time,volts,Loc,time_err,0,2,0,0)

        %Plot_data(2,0,'g',Raw_data,-1,2+1)

        %Plot_data(2,0,'r',Raw_data,-2,2+2)

        %Plot_data(2,0,'c',Raw_data,-3,2+3)

        Plot_data(2,0,'m',time,volts,Loc,time_err,-0.75,6,0,0)


        % Data is overlaid to see if the phasing is correct

        Plot_data(3,1,'b',time,PR,Loc,time_err,0,1,0,0)

        Plot_data(3,0,'g',time,PR,Loc,time_err,0,9,0,0)

        Plot_data(3,0,'r',time,PR,Loc,time_err,0,2,0,0)

        Plot_data(3,0,'c',time,PR,Loc,time_err,0,3,0,0)

        Plot_data(3,0,'m',time,PR,Loc,time_err,0,4,0,0)

        Plot_data(3,0,'y',time,PR,Loc,time_err,0,5,0,0)
```

```
Plot_data(3,0,'k',time,PR,Loc,time_err,0,6,0,0)


% A contour plot is plotted and a file outputed
figure(5); close; figure(5)
[min(min(PR_Z)) max(max(PR_Z))]
contourf(PR_X,PR_Y,PR_Z,[0.4752:(1.4619-0.4752)/35:1.4619])
%contourf(PR_X,PR_Y,PR_Z,35)
shading flat
axis equal
temp = axis;
axis([-.5 1 temp(3) temp(4)])


% Single blade passage is plotted
figure(7); close; figure(7)
temp_X = PR_X(1:2*floor(size(PR_X,1)/22),:);
temp_Y = PR_Y(1:2*floor(size(PR_X,1)/22),:);
temp_Z = PR_Z(1:2*floor(size(PR_X,1)/22),:);
contourf(temp_X,temp_Y,temp_Z,75)
shading flat
axis equal


%print fred -depsc2
```

mov_avg.m

```
% m-function file to calculate the moving averages at a particular point
% x,y data
% points ahead and behind central one ie 0 return same data, 1 = 3 points, 2 = 5
points
% n, polynomial to fit, 0 = average, 1 = linear, 2 = parabolic etc


function [y_avg] = mov_avg(x,y,points,N)


y_avg = y;
```

```
        x_poly = zeros(1,(2*points+1));

        y_poly = zeros(1,(2*points+1));


        for i = (points+1):((length(x)-points)-1)

            for j = -points:points

                x_poly(j+points+1) = x(i+j);

                y_poly(j+points+1) = y(i+j);

            end

            y_avg(i) = mean([max(y_poly) min(y_poly)]); % Most effective method, just take
the mean of the max and min

        end


        % Leading points

        if points > 0

            % Leading few points

            for j = 1:(2*points+1)

                y_poly(j) = y(j);

            end


            % Leading moving average

            for i = 1:points

                y_avg(i)  = mean([max(y_poly) min(y_poly)]);

            end % for i = 1:points


            % Trailing few points

            for j = (2*points+1):-1:1

                y_poly(j) = y(length(x)-j+1);

            end


            % Trailing moving average

            for i = (length(x)-points):length(x)

                y_avg(i)  = mean([max(y_poly) min(y_poly)]);
```

```
        end % for i = 1:points
    end % if points > 0
```

phase_data.m

```
    % m-function file to phase the Kulite data into a single line

    function[time,volts] = phase_data(time,volts);

    % Constants are read in
    [Kul_order,Kul_offset,Blade_no,Blade_th,gam_gas,Kul_ax_cho,Diameter,Chord,shock,SC
F] = Kulite_constants_stall_90;

    % A matrix of the time vector is made
    time = time*ones(1,size(volts,2));

    % Offset to ensure that each Kulite gives data from the same blade
    for i = 1:length(Kul_order)
        time(:,Kul_order(i)) = time(:,Kul_order(i))-Kul_offset(i);
    end % for i = Kul_order

    % The stall cell moves slower than the RPM.
    M   = Blade_no - (1/SCF)*Blade_no; % Number of blade passages that stall cell
moves per revolution
    M_N = M/Blade_no;              % Fraction of a blade that the stall cell moves per
pitch

    % A different offset is needed to make the data from the same position within the
stall cell
    for i = 1:length(Kul_order)
        % Fraction of offset of stall cell
        temp  = M_N*Kul_offset(i);
        fract = temp-floor(temp); % Fraction between blades
        %[time(1,Kul_order(i)) time(end,Kul_order(i))]
        %[i Kul_offset(i) temp floor(temp) temp-floor(temp) ceil(temp)]
```

```
            % Temporary floor and ceiling time
            time_floor  = time(:,Kul_order(i))+floor(temp);
            time_ceil   = time(:,Kul_order(i))+ceil(temp);

            % Intepolation of time before the stall cell
            %I               =find(time(:,Kul_order(i))>time(1,Kul_order(i))+ceil(temp));

            I               =
find(time(:,Kul_order(i))<(time(end,Kul_order(i))+floor(temp))); % There is an offset
            volts_floor        = zeros(size(time(:,Kul_order(i))));
            volts_floor(I(end):end) = volts(I(end):end,Kul_order(i));
            volts_floor(1:I(end))   =
interp1(time_floor,volts(:,Kul_order(i)),time(1:I(end),Kul_order(i)),'cubic');

            % Intepolation of time after the stall cell
            volts_ceil        = zeros(size(time(:,Kul_order(i))));
            volts_ceil        = volts(:,Kul_order(i));
            volts_ceil(1:I(end)) =
interp1(time_ceil,volts(:,Kul_order(i)),time(1:I(end),Kul_order(i)),'cubic');

            % Final interpolation trying to match the speed of the stall cell
            %I = find(time(:,Kul_order)>time(1,Kul_order(i))+ceil(temp));
            I = find(time(:,Kul_order(i))<(time(end,Kul_order(i))+floor(temp))); % There
is an offset
            volts(1:I(end),Kul_order(i)) = fract*volts_floor(1:I(end)) + (1-
fract)*volts_ceil(1:I(end)); %Thought to be correct
            %volts(1:I(end),Kul_order(i)) = (1-fract)*volts_floor(1:I(end)) +
(fract)*volts_ceil(1:I(end)); % Cowboy thing again

            if 0
                figure(6); close; figure(6);
                %plot(time_floor,volts(:,Kul_order(i)),'r')
                plot(time(:,Kul_order(i)),volts_floor,'r')
                hold on
                %plot(time_ceil,volts(:,Kul_order(i)),'g')
```

```
        plot(time(:,Kul_order(i)),volts_ceil,'g')
        plot(time(:,Kul_order(i)),volts(:,Kul_order(i)),'b')
    end % if 0

end % for i = 1:length(Kul_order)
```

Plot_data.m

```
% m-file to plot a certain time part of the stall data file

function [fred] =
Plot_data(fig_no,bool_new_fig,fig_colour,time,volts,Loc,time_err,offset,Data_column,...
    subplot_no,subplot_tot)

% Time period is defined
time_start = time(1,Data_column);   % Start time of sample
time_end   = time(end,Data_column); % End time of sample
%time_start = 16.3           % User defined start time
%time_end   = 17             % User defined end time

% Time period entry points are found
temp = find(time(:,Data_column)>time_start & time(:,Data_column)<time_end);

% figure is plotted and trimmed
if bool_new_fig
    figure(fig_no); close; figure(fig_no);
else
    figure(fig_no);
end

if subplot_no ~= 0
    subplot(subplot_tot,1,subplot_no)
end % if subplot_no ~= 0

plot(time(temp,Data_column),volts(temp,Data_column)+offset,fig_colour)
```

```
hold on
xlabel('Time [s]'); ylabel('Raw Voltage signal [V]')
grid on

temp = axis;
h = line([time(Loc(1:end),Data_column)
time(Loc(1:end),Data_column)]',(ones(size(Loc(1:end)))*([temp(3) temp(4)]))');
set(h,'Color',[0 0 0]);  % Makes colour of line black
h = line([time(Loc(2:end),Data_column)-time_err time(Loc(2:end),Data_column)-
time_err]',(ones(size(Loc(2:end)))*([temp(3) temp(4)]))');
set(h,'Color',[0 0 0]);  % Makes colour of line black

%axis([temp(1) temp(2) 0 0.7])

if subplot_no ~= 0
    axis([0 120 0.5 1.3])
end % if subplot_no ~= 0
```

Plot_rpm.m

```
% mfile to plot RPM through the stall

function [fred] = Plot_rpm(fig_no,RPM,RPM_Hz)

%temp = find(Raw_data(:,9)>0);

figure(fig_no);% close; figure(fig_no);
%plot(Raw_data(temp,1),Raw_data(temp,9))
%plot(RPM(:,1),RPM(:,2),'k');
plot(1:length(RPM(:,1)),RPM(:,2)/RPM_Hz,'k');
hold on
xlabel('Revolutions'); ylabel('rpm'); title('RPM through stall at 100% speed')
```

Process_data.m

```
% m function file to organise the data of the Raw data file

function [time,volts,tach,Loc,Hz,time_err] = Process_data(Raw_data,N_mov_avg)

% Data is sorted into simpler to use groups
time    = Raw_data(:,1);    % Kulite time data
volts   = Raw_data(:,2:10); % Kulite raw voltage data
tach    = Raw_data(:,12);   % Kulite trigger voltage
samples = length(time);     % Number of samples

% Trigger level is calculated
Trig = mean([min(tach) max(tach)]);

% Location of trigger points and correction to exact trigger timing point
Loc = find( tach(2:samples)<Trig & tach(1:samples-1)>Trig ); %location of time of
start of rev
Hz = (length(Loc)-1)/(time(Loc(end))-time(Loc(1))); % Frequency of rotor
revolution over the sample period

% If Loc (location) if at the beggining of the sample it is discarded
if Loc(1) < 3
    Loc = Loc(2:end);
end

% Last trigger is discarded to ensure trailing zeros resulting from probe lining
up do not effect the calculations.
Loc = Loc(1:(end-1));

% Timing correction to ensure that each sample begins at the correct time
d_time_Loc(:,1) = time(Loc+1)-time(Loc);  % Time interval between trigger and next
time interval
```

```
        d_tach_Loc(:,1) = tach(Loc+1)-tach(Loc);  % Ramp slope between trigger and next
time interval
        m               = d_tach_Loc./d_time_Loc; % Slope
        c               = tach(Loc);              % Intercept
        time_err        = (Trig - c)./m;          % Error in trigger timing

        % Error in trigger timing is converted to be directly subtracted from the period
        time_err = (time_err(2:end)-time_err(1:end-1));

        % The RPM based on each trigger is calculated
        Hz = time(Loc(2:end)) - time(Loc(1:end-1));
        Hz = Hz + time_err;
        %RPM = 60./RPM;

        Hz_old = Hz;
        Hz     = [time(Loc(2:end)) Hz]; % First column is time signal and the second is
the Hz

        % Data is moving averaged a few times to remove the wiggle
        for i = 1:N_mov_avg
            [Hz(:,2)] = mov_avg(Hz(:,1),Hz(:,2),1,1);
        end % for i = 1:,N_mov_avg
        %[Hz(:,2)] = mov_avg(Hz(:,1),Hz(:,2),1,1);
        %[Hz(:,2)] = mov_avg(Hz(:,1),Hz(:,2),1,1);
        %[Hz(:,2)] = mov_avg(Hz(:,1),Hz(:,2),1,1);

        % This is to correct the RPM
        time_err = time_err + (Hz(:,2)-Hz_old);
        %[1./Hz_old 1./Hz(:,2) 1./(time(Loc(2:end))-time(Loc(1:end-1))+time_err)]

        % Data is converted to Hz or RPM
        Hz(:,2) = 1./Hz(:,2);
```

Time_to_pitch.m

```
% m-function file to normailise the Kulite data from time to physical domain

% each blade pitch is considered to be unity


function [time_norm] = time_to_pitch(time,Loc,time_err)


[fred,fred,Blade_no,fred] = Kulite_constants_stall_90;


% Tempory time variable vs pitch is set up as the trigger position is known

temp_time  = [time(Loc(1)); time(Loc(2:end))-time_err];

temp_pitch = Blade_no*(0:(length(temp_time)-1))';


% Interpolation along the time is performed

time_norm = interp1(temp_time,temp_pitch,time,'cubic','extrap');
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Sanger, N.L., “Design of Low Aspect Ratio Transonic Compressor Stage Using CFD Techniques,” ASME Journal of Turbomachinery, July 1996, Vol. 118 pp 479 – 491.

2. Sanger, N.L., “*Design Methodology for the NPS Transonic Compressor*,” TPL Technical Note 99-01, Naval Postgraduate School, Monterey, California, August 1999.

3. Gannon, A.J., Hobson, G.V. and Shreeve ,R.P., 2004, “A Transonic Compressor Stage Part 1: Experimental Results, “ASME GT2004-53923, Turbo Expo,Vienna, Austria.

4. Gannon, A.J., Hobson, G.V. and Shreeve ,R.P., “Measurement of the Unsteady Casewall Pressures Over a Rotor of a Transonic Fan and Comparison with Numerical Predictions”, ISABE 2005, 17th International Symposium on Airbreathing Engines, Munich, September 2005.

5. Rodgers, M. W. Unsteady Pressure Measurements on the Case Wall of a Transonic Compressor, Master’s Thesis, Naval Postgraduate School, Monterey, California, June 2003.

6. Villescas, I., Flow Field Surveys In a Transonic Compressor Prior To Inlet Steam Ingestion Tests, Master’s Thesis, Naval Postgraduate School, Monterey, California, September 2005.

7. Brunner, M.D., Experimental and Computational Investigation of Flow in a Transonic Compressor Inlet, Master’s Thesis, Naval Postgraduate School, Monterey, California, September 2005.

8. Payne, T.A., Inlet Flow-Field Measurements of a Transonic Compressor Rotor Prior to and During Steam-Induced Rotating Stall, Master’s Thesis, Naval Postgraduate School, Monterey, California, December 2005.

9. Sussman Automatic Corporation. “SVS600 Electric Steam Boiler Specifications.”, 2004, Long Island City, New York.

10. Hewlett-Packard Operating Manual, HP E1422A Remote Channel Multi-function DAC Module, Edition 1, E1422-90000, May 1999.

11. Erwin, J. R., A Review of the Design of the NPS/TPL Transonic Compressor, Contractor Report No. NPS67-83-004CR, Naval Postgraduate School, Monterey, California, 1983.

12. Hewlett-Packard Operating Manual, HP E1422A Remote Channel Multi-function DAC Module, Edition 1, E1422-90000, May 1999.

13. HP VEE Pro (version 6.01), August 2000.

14. Hewlett-Packard Operating Manual, HP E8402A, E8404A C-Size Mainframes, Edition 1, E8402-90000, May 1998.

15. Agilent Technologies Operating Manual, N2216A VXI/SCSI Interface Module, N2216-90001, July 2000.

16. HP DAC Express (version 2.01), September 2000.

17. MATLAB (version 5.3.0 R11), January 1999.

18. Hobson, G.V., (Unpublished) Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, Calfornia, May 2003.

19. Cumpsty, N.A., Jet Propulstion: A simple guide to the aerodynamic and thermodynamic design and performance of jet engines ,Cambridge University Press, New York, 1997.

20. Gannon, A.J., Hobson, G.V. and Shreeve ,R.P., "Experimental Investigation During Stall and Surge in a Transonic Fan Stage and Rotor-Only Configuration", TURBO EXPO 2006, ASME TURBO EXPO 2006: Land, Sea and Air, Barcelona, Spain, May 2006.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Distinguished Professor and Chairman Anthony Healey
   Department of Mechanical and Aeronautical Engineering
   Naval Postgraduate School
   Monterey, California

4. Professor Garth Hobson
   Department of Mechanical and Aeronautical Engineering
   Naval Postgraduate School
   Monterey, California

5. Dr. Anthony Gannon
   Department of Mechanical and Aeronautical Engineering
   Naval Postgraduate School
   Monterey, California

6. Naval Air Warfare Center
   Propulsion and Power Engineering
   ATTN: Mark Klien
   Patuxent River, Maryland

7. ENS William Levis
   Monterey, California